Consiglio Nazionale delle Ricerche

Aspects of Modeling and Verifying Secure Procedures

M. Petrocchi

IIT TR-24/2005

Technical Report

Novembre 2005

Istituto di Informatica e Telematica

UNIVERSITÀ DEGLI STUDI DI PISA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DOTTORATO DI RICERCA IN
INGEGNERIA DELL'INFORMAZIONE

Ph.D. Thesis

Aspects of Modeling and Verifying Secure Procedures

MARINELLA PETROCCHI

February, 2005

Advisors:
Prof. Nicoletta De Francesco
Dr. Anna Vaccarelli
Prof. Gigliola Vaglini

Abstract

Security protocols are specifications for exchanging messages on a possibly insecure network. They aim at achieving some security goals (*e.g.*, authenticating the parties involved in a communication, or preserving confidentiality of certain messages) preventing some malicious party to achieve advantages for its own. Goals of security protocols are generally achieved through the use of cryptography, the art of writing in secret characters, not comprehensible to anyone but the sender and the intended recipient.

There is however a branch, in the computer science community, that, among its wide field of activities, aims at studying possible attacks on secure procedures without breaking cryptography, *e.g.*, by manipulating some of the exchanged messages. This is the *formal methods* community, with an eye for security.

This thesis mainly investigates the formal modeling and analysis of security protocols, both with finite and non finite behaviour, both within a process-algebraic and an automata framework. Real life protocols for signing and protecting digital contents and for giving assurance about authentic correspondences will be specified by means of the above cited formalisms, and some of their properties will be verified by means of formal proofs and automated tools.

The original contributions of this thesis are the following. Within the framework of a formal modeling and verification of security protocols, we have applied an automated tool to better understand some secure mechanisms for the delivery of electronic documents. This has given us a deep insight on revealing the effects of omitted (or even erroneously implemented) security checks. Furthermore, a formal framework for modeling and analysing secure multicast and wireless communication protocols has been proposed. The analysis is mostly based on some new compositional principles giving sufficient conditions for safely composing an arbitrary number of components within a unique system. Also, steps towards providing the Team Automata formalism (TA) with a framework for security analysis have been taken. Within the framework, we model and analyse integrity and privacy properties, contributing to testify the expressive power and modelling capabilities of TA.

To my parents, Olinto and Maria Grazia
Ai miei genitori, Olinto e Maria Grazia

Acknowledgments

Most of this work of thesis would not have been possible without the advice of Fabio Martinelli and of my advisors, Nicoletta De Francesco, Anna Vaccarelli and Gigliola Vaglini.

Also, here is a list of people I wish to thank. The list is randomly sorted. I hope that, by finding your name in the list, you will know why you are here.

Luca Durante, Riccardo Focardi, Gabriele Lenzini, Maurice ter Beek, Lavinia Egidi, with Federico and Domitilla, Franco Denoth, Chiara Braghin, Maria Grazia Vigliotti, Marta Simeoni, Ivano Salvo, Paolo Mori, Salvatore Minutoli, Gianluigi Zavattaro, Andreas Pfitzmann, Sandro Etalle, Rosa Meo, Roberto Gorrieri, Beatrice Lami, Claudio Baesso, Ada De Giorgio, Adrianna Alexander, Massimo Bartoletti, Giacomo Terreni, Stefano Zacchiroli, Giovanni Stea, Cinzia Bernardeschi.

Carlo and Walter Bruno, with Enzo and Maria Consiglia, Elena Michelucci, Alberta Schiavon, Fabrizio Tocchini, Maria Giovanna Melloni, Benedetta Calamari, Elena Petrocchi, Zrinka Rezic.

Pietro Volpi, Marianna Bertini, Mario and Lebda Petrocchi.

Lavinia Egidi and Paolo Mori win a second mention, for the suggestion of substantial improvements to the overall presentation.

Special thanks go to Giovanni.
Un ringraziamento speciale a Giovanni. Grazie mille.

Finally, congratulations to myself for all I have managed to do, for better or for worse. Hoping not to end up like all those English people who every year throw themselves into the Thames because they did not play their trumps. :-)
Infine, mi congratulo con me stessa, per quel che son riuscita a fare, nel bene e nel male. Con l'augurio di non far la fine di tutti quegli Inglesi che ogni anno si buttano nel Tamigi per non aver battuto le atout. :-)

Contents

List of Figures

Chapter 1

Introduction

In computer science, a protocol is a set of rules, or procedures, for transmitting data between electronic devices, such as computers. In particular, a security protocol is a specification for transmitting those data in a safe way, *e.g.*, avoiding them to be unveiled to unauthorized users or to be modified during their journey from the sender to the intended recipient. These and other goals are generally achieved through the use of cryptography, *i.e.*, the art of writing in secret characters, not comprehensible by anyone but the authorized parties. There is however a branch, in the computer science community, that, among its wide field of activities, aims at studying possible attacks on secure procedures without breaking cryptography, *e.g.*, by manipulating some of the exchanged messages. For years, the formal methods (and security) community has been working in these topics. This work of thesis is all based on the same thread, *i.e.*, to give methodologies for the modeling and analysis of security protocols, by considering cryptography reliable.

1.1 Formal methods and security

The use of cryptographic primitives is by now a standard practice within the area of Internet communication. In the last decades, many researchers have covered various mathematical issues involved in these primitives, leading to a better understanding of the foundations of cryptography. On the other hand, using such well understood cryptographic primitives does not give full guarantees about the fulfillment of the required security properties. Indeed, threats can lie, for example, in the way messages are exchanged over the network.

A simple example is here informally presented, to show the possible uncertainty about the correctness of a protocol specification. Notation is quite intuitive. However, the reader is invited to see Chapter 2 for details about notation and cryptographic constructs.

> **A simple example.** Let the reader suppose that A wants to send to the bank $Bank$ an order to move some money to X's account. Thus, A sends the message "*move $1000 to X's account*", signed with its private key, denoted by pk_A^{-1}. (We anticipate here that a digital signature is a cryptographic construct aiming at assuring authentication of origin and integrity to a message.)
>
> $$A \mapsto Bank : \{\text{move \$1000 to X's account}\}_{pk_A^{-1}}$$
>
> Since this message is signed by pk_A^{-1}, $Bank$ should be assured that it has been originated by A. Thus, $Bank$ makes the money transfer. Now, let the reader suppose that X eavesdrops on this message. It can pretend to be A and it can send the message again to $Bank$, *i.e.*, :
>
> $$X(A) \mapsto Bank : \{\text{move \$1000 to X's account}\}_{pk_A^{-1}}$$
>
> The signature of this message is still valid. Possibly, X gets $2000. Thus, the protocol has been attacked, even without breaking cryptography.

Starting from these observations, a branch of research in computer security assumes cryptographic primitives to be perfect, and uses a black box view of cryptography.

Within this last area, formal methods and tools have been successfully applied for the analysis of network security. Exploiting formal methods, the protocol under investigation is described in a given language, then a formal specification of the security property to be analyzed is defined. Whether or not the security property is fulfilled is investigated by formally analyzing the protocol within a hostile environment, *i.e.*, considering the presence of a malicious agent running the protocol together with the honest participants.

In many occasions, formal methods have been proved efficient either to better define the goals of a security protocol (and the mechanisms through which they are achieved) and to offer a rigorous description of the interactions among the

participants. Indeed, many examples of correctness of a security protocol –as well as the discovery of attacks on them– may be found in the literature, *e.g.*, [AG97, FGM00a, MM99, Mea95, LR97, SS98, FJTG99]. Furthermore, several formal techniques have been developed. Some are based on state exploration, and they can typically ensure error-freeness for bounded systems (*e.g.*, [Low96, MMS97, RG97]). Other approaches are based on proof techniques for authentication logic (*e.g.*, [AT91, KW96, Pau97]). Type systems and other static analyses have also been successfully exploited (*e.g.*, [Aba99, BDNN01]).

In the following, we give a paradigmatic example of how errors can be found in security protocols using a process algebra based formalism and an analysis tool.

1.1.1 The Needham-Schroeder public key protocol

The aim of the Needham-Schroeder protocol [NS78] is to establish mutual authentication between two users A and B . It uses public key cryptography and nonces, *i.e.*, parameters that vary with time, and generated with the purpose of being used in a single run of the protocol, [Low95], (more details in Chapter 2, Section 2.1).

$$
\begin{array}{lllll}
1 & A \mapsto AS & : & A, B \\
2 & AS \mapsto A & : & \{B, pk_B\}_{pk_{AS}^{-1}} \\
3 & A \mapsto B & : & \{A, n_A\}_{pk_B} \\
4 & B \mapsto AS & : & B, A \\
5 & AS \mapsto B & : & \{A, pk_A\}_{pk_{AS}^{-1}} \\
6 & B \mapsto A & : & \{n_A, n_B\}_{pk_A} \\
7 & A \mapsto B & : & \{n_B\}_{pk_B}
\end{array}
$$

The protocol starts with A consulting the authentication server AS in order to obtain B 's public key (step 1). In step 2 AS replies to A by signing the public key of B. It is assumed that A knows the public key of the authentication server, in order to verify the signature. Communication with B starts in step 3, where the message is encrypted by B's public key. Thus, only B can decrypt it. This message means that someone who claims to be A wishes to establish communication; n_A is the nonce generated by A . Upon decrypting the message, B asks for A's public key to AS (step 4). AS replies in step 5, similarly to what replied to A in step 2. At this point a double handshake is needed to authenticate A and B to each other. In step 6, B replies to A, sending the the new nonce n_B and the one received from A (n_A), both encrypted with A 's public key. When A receives the nonce $n_A$back, it can conclude that is really talking with B since only B could have decrypted the

message sent by A containing n_A. In the second message A replies to B , sending back the n_B nonce. Following the same reasoning as above, B will conclude that he is indeed talking with A .

The protocol is composed by seven steps, but four of them can be avoided if A and B have local caches of commonly used public key.

In 1995, a good 17 years after the publication of this protocol, Gavin Lowe found the following attack, [Low95, Low96]. An intruder X may impersonate A, by inciting B to initiate a second session. The messages involved in the attack are the ones of the interactions between A and B. Thus, we ignore the exchanges with the authentication server AS. Also, it is assumed that the intruder X possesses a key pair pk_X, pk_X^{-1} and that the public keys at stake are known by everybody (as is common in security analysis, public objects are assumed to be available to everybody). We indicate with *i* steps of the first session and with *ii* steps of the second session. The first session sees A as the initiator of the protocol and X as the responder, whereas the second session sees X as initiator and B as the responder.

$$\begin{array}{lll} 3.i & A \mapsto X & : \{A, n_A\}_{pk_X} \\ 3.ii & X(A) \mapsto B & : \{A, n_A\}_{pk_X} \\ 6.ii & B \mapsto X(A) & : \{n_A, n_B\}_{pk_A} \\ 6.i & X \mapsto A & : \{n_A, n_B\}_{pk_A} \\ 7.i & A \mapsto X & : \{n_B\}_{pk_X} \\ 7.ii & X(A) \mapsto B & : \{n_B\}_{pk_B} \end{array}$$

In step 3.ii X pretends to be A and initiates a session with B, by replying the message sent by A in step 3.i. As a consequence of the interleaving of the two sessions, with the messages opportunely replied, X eventually discovers n_B, with the help of an unaware A. B concludes that it is indeed talking with A, when, instead, it is talking with X.

The attack was found by running FDR, a model checker for formal automated analysis, on a process algebra specification of the protocol. Lowe subsequently proposed to fix the protocol by including the respondent's identity in step 3. By running the corrected specification with the same tool, he did not find any attack.

1.2 Our line of research

Given the (possible) sensitive nature of the information exchanged in a communication protocol, it appears reasonable to think about the presence of an adversary,

either passive or active, able to interact (by eavesdropping on and by manipulating messages) with the honest participants of a protocol, in order to achieve advantages for its own. Indeed, the above-presented case study is a notable example: the adversary, by opportunely following the steps of the protocol, is able to cheat party B, since at the end of the second session party B believes to have talked to party A. Furthermore, X discovers n_B. If n_A and n_B are used as authenticators, X has the ability to impersonate A to B for the rest of the session. Finally, this leads to an inconsistent state, in which B believes that A has initiated a communication with it, when in fact it has not.

Of course, this is not the unique example of the underlying uncertainty about the good and the evil in a security protocol specifications. A lot of significant work has been done in the past years for formally developing techniques and tools for modeling and analysis of protocols (see, *e.g.*, references in Section 1.1).

However, fields of survey are wide in this area and not all the formal and security aspects were exhaustively investigated. In particular, we decided to investigate the following three topics.

- The modeling and analysis of secure multicast/wireless protocols. Motivations for this research should be found in the diversity of such protocols from standard cryptographic schemes. Indeed, these protocols rely on an underlying infinity, since a continuous (and possibly unbounded) stream of messages is sent to a possibly unbounded set of receivers. Furthermore, particular scenarios may include mobility. Such features make these protocols unfeasible to be verified with standard tools such as model checkers (unless their specification is refined, see the debate between [Arc02] and [BL02]). We anticipate that our approach is quite different from what has been proposed in the literature and focuses its attention on the verifiability of a system with an arbitrary number of components.

- The modeling and analysis of secure procedures for the delivery of digital certificates. Motivations for this research should be found in the need for a careful investigation of the aims and security mechanisms suggested by some documents, like Request for Comments (RFCs) and Internet Drafts, outlining the guidelines for building up a secure architecture. Often, documents of this kind are difficult to understand in their entirety. This is mainly due to the informal way in which assumptions, requirements and goals of a procedure are literally described. We anticipate that our approach focuses on a translation of a series of informal steps into a given formalism and an

evaluation of the consequences of the lack of some security mechanisms in the specifications.

- The provision for Team Automata of an equipment for modeling security protocols and analysing some of their properties. Originally introduced in the context of Computer Supported Cooperative Work (CSCW) [BEKR03, Ell97, Kle03], they have been proved to form a flexible framework for modelling communication between components of distributed and reactive systems. Thus, motivations for our research should be found in the will to extend the use of this formalism with capabilities to treat also security aspects of distributed and reactive systems. We anticipate that our approach shows benefits in the natural way in which we model various kind of communication (*e.g.*, the multicast one) and in the feasibility of performing the analysis of some security properties, among them *privacy*, that is considered a very important topic nowadays.

1.3 Contributions

With respect to the formal modeling and analysis of security protocols, the thesis gives the following main contributions.

To the best of our knowledge, the first modeling of various real life secure procedures is given in the thesis. Starting from protocols to authenticate data streams, passing through the world of secure mobile agents, not to mention commercially available protocols for the secure delivery of electronic documents, both a process-algebraic framework and an automata framework are exploited to highlight their expressiveness in specifying such systems.

Compositionality principles for safely composing processes, without loosing the security properties that each single process enjoys before the composition, are first defined, then formally proved and finally applied to some case studies, for verifying some of their security properties. Again, results are shown both in a process-algebraic framework and within Team Automata (TA).

As a step towards a comprehension of the potentialities of exploiting formal methods, and of the intrinsic limitations of informally specifying security procedures, an automated verification tool is used to analyze the secure and correct emission of a digital certificate. The analysis gives an insight on the possible consequences of the omission of some security mechanisms (or of their erroneous implementation).

Finally, the investigation of the possible equipment of TA with a framework for treating security aspects leads to the following results. First, an insecure communication scenario for TA is defined. Secondly, the Generalized Non-Deducibility on Compositions (GNDC) schema of [FM99] is reformulated in terms of TA and, subsequently, a compositional analysis strategy is described for it. Then, a privacy property is analyzed, and this represents the first attempt to use TA for modeling privacy. To conclude, we show that TA capture in a native way the one-to-many and one-to-all communications that are so typical of multicast and broadcast communications.

1.3.1 Bibliographical note

The work we present here has been already published in international conferences. In particular:

- The modelling and analysis of secure multicast in the process-algebraic framework appeared in [GMPV03a, GMPV03b, MPV03].
- Results related to Team Automata were published as [BLP04a, BLP04b, BLP03, EP04].
- The work on digital certificates has been divulged to the scientific community through [GMPV01, MPV02].
- Additional material related to mobile computing appeared as [DP03] (a secure protocol) and [MPV04] (on selfish behaviour in ad hoc networks).

We acknowledge joint research with Maurice ter Beek, Nicoletta De Francesco, Lavinia Egidi, Roberto Gorrieri, Gabriele Lenzini, Fabio Martinelli and Anna Vaccarelli.

1.4 Outline of the work

The original contributions of the thesis are organized into three main chapters, plus three appendixes. Each of them (with exclusion of Appendix C), begins with an introduction, that describes its contents and its main results. Related work in the area are generally embedded into the chapter (or appendix).

The thesis is mainly about formal modeling and analysis of security protocols, both with finite and not finite behaviour, both from a process algebra point of view and from a team automata point of view.

We describe the work presenting both modeling and analysis in the main chapters, whereas the first two appendixes are devoted to describe work about architectures of systems (*i.e.*, an original architecture is precisely proposed).

Thus, the reminder of the thesis is as follows.

Chapter 2 introduces some general background, aimed at helping the reader in acquiring the basic notions required for the following chapters. More precisely, security properties and security primitives are friendly introduced. Secondly, the informal notation with which security protocols are generally described in the literature is presented. Then, hints to the two formalisms mainly used throughout this work are introduced. In particular, details are given about Team Automata and about two process algebras for the specification of cryptographic protocols. Finally, Section 2.4 reminds the reader of two well-known schemes for the definition and analysis of (timed) security properties.

In the three main chapters, a journey is done towards the modeling and analysis of some security procedures. More precisely:

- Chapter 3, also known as the *Multicast* chapter, is devoted to modeling and analysis of multicast protocols. It first describes relevant multicast security protocols. Then, new analysis strategies are presented. Finally, we analyze two of the modelled protocols, by means of the above-cited strategies. The *multicast* chapter contributes towards the analysis of multicast security protocols (with a non finite behaviour with respect to the number of participants). A process-algebraic framework is adopted.

- Chapter 4, also known as the *Team Automata* chapter, is devoted to modeling and analysis in the framework of TA. A first part describes a relevant multicast security protocol and a secure protocol for mobile agents, by means of TA. New analysis strategies within TA are then presented. Finally, we analyze the modelled protocols, by means of the above-cited strategies. The *Team Automata* chapter contributes towards the analysis of procedures with non finite behaviour with respect to the number of participants. Furthermore, it gives the first attempt to analyze security and privacy properties by TA. An automaton framework is adopted.

- Chapter 5, also known as the *Digital Certificate* chapter, introduces some basic notions on the automatic verification of finite-state security proce-

dures. Then, part of the chapter is devoted to formally describe real world procedures for digital certificates enrollment and delivery. The procedures are hereafter analyzed by means of a software tool and the results are reported in the chapter. The *digital certificate* chapter contributes towards the automatic analysis of procedures with finite behaviour. Furthermore, it gives a better understanding of how the lack of certain security checks may result in breaking the desired properties of a specification. A process-algebraic framework is adopted.

Chapter 6 summarizes contents and results of the three main chapters.

In the appendixes, two architectures are shown, to model a secure multicast protocol and a scheme to fight against a new threat in mobile networks. More precisely:

- Appendix A introduces a general architecture for giving authenticity to a pre-existent reliable protocol for mobile computing.
- Appendix B introduces a scheme for monitoring possible selfish behaviors in mobile ad hoc networks.

Finally, Appendix C shows excerpts of the formal specifications informally described in Chapter 5.

Chapter 2

Preliminaries

In this chapter we recall the basic cryptographic and modelling tools that will be used throughout the thesis, for the sake of completeness, and in order to fix terminology and notation.

2.1 Security

We give here some basic (and) informal notions of the three most common security properties. They will be more formally treated and rephrased throughout the thesis.

- *Secrecy* (*i.e.*, confidentiality of the exchanged messages) is that property that should ensure the protection of information against unauthorized disclosure.
- *Integrity* (*i.e.*, no alteration of the content of a message, also known as *message authenticity*) is that property that should ensure the protection of information against unauthorized modification.
- *Authentication* (*i.e.*, capability of correctly identifying other parties during a communication, also known as *entity authentication*) is that property that should ensure the protection of the identities of the parties against unauthorized modification.

2.1.1 Cryptographic primitives

Symmetric cryptography, also called secret key cryptography, involves the use of a secret key known only to the participants of the secure communication. If Alice

wants to send the message securely over a public channel to Bob, she uses the key they agreed on before, to send to Bob. He will decrypt the received cyphertext with the same key to gain access to the message.

Asymmetric cryptography, also called public key cryptography, involves the use of a pair of keys. Each pair consists of a public key and a private key. Alice and Bob holds their own pair. What is encoded with one key, can only be decoded with the other. As the terminology suggests, the private key remains a closely guarded secret of its owner, whereas the public key is published so that everybody knows it.

One can achieve both secrecy and authentication of origin by means of public key cryptography. Indeed,

- to achieve secrecy, Alice encrypts message m with Bob's public key; Bob decrypts the cyphertext it with his private key. Secrecy is guaranteed since only Bob knows his private key and thus only he could decrypt the cyphertext.

- to achieve authentication of origin, Alice encrypts message m with her private key; Bob decrypts the cyphertext with Alice's public key. Authentication of origin is guaranteed since only Alice knows her private key and thus only she could have generated the cyphertext.

Other cryptographic constructs useful throughout the thesis are the following.

- One-way hash functions, a class of mathematical functions with the following peculiarities: i) they map arbitrarily long binary strings into strings of a fixed length; ii) they are "collision resistant", *i.e.*, only with negligible probability it is possible to obtain the same output from two different inputs; iii) they are not reversible (at least, with high probability).

  We refer to the fixed length output of an hash functions as the *fingerprint* of the input message.

  The functions are public, *i.e.*, no secrets are involved in computing a one-way hash. They are used to check for matches, without discovering the data that are being compared. Let us suppose that Bob and Alice receive a catalog and want to know if they are qualified for the same discount. They can compute the hash of the catalog they have received, and compare the fingerprints. If they do not match, there something in Alice's catalog that does not match what is in Bob's catalog.

- Message Authentication Codes (MACs) are used to verify the authenticity of a message. Suppose that Alice (the sender of a message) and Bob (the recipient) share a secret key. Alice uses the message and the key to compute the MAC, and sends the MAC along with the message. When Bob receives the message, he computes the MAC, and then checks to see if his MAC matches Alice's. If it does, then he knows the message is from Alice and that nobody has changed it since she sent it.

  If an adversary does not have the secret key, then even though he is able to modify the message, he cannot produce the matching MAC. Therefore Bob will detect the alteration.

- Digital signatures (*e.g.*, [RSA78]) are electronic signatures that can be used to authenticate the identity of the sender of a message or the signer of a document, and possibly to ensure that the original content of the message or document that has been sent is unchanged. Furthermore, the ability to ensure that the original signed message arrived means that the sender cannot easily repudiate it later.

  In practice, a digital signature is the result of the application of a private key to the fingerprint of a document. Suppose Alice wants to digitally sign document d. First, she computes the hash on that document. Then, she encrypts the fingerprint with her private key. Finally, she sends both the document and the signature to Bob. On his hand, Bob hashes the received document. Then, he uses Alice's public key to decrypt the fingerprint of the document. Finally, he compares the fingerprints. If they match, the signature is valid.

- Digital certificates, [HFPS99], that are electronic documents linking an identity (*i.e.*, a person or a machine) to a public key. They are issued by a Certification Authority that can vouch for an individual identity. The way CA vouches for such links is to digitally sign the issued certificate with CA's private key. Typically, a digital certificate contains a public key, information specific to the user (a name, a company, an IP address, etc.), information specific to the Certification Authority issuer, a validity period (starting date - finishing date) and additional management information.

- Nonces. In information technology, a nonce is a parameter that varies with time. A nonce can be a time stamp, a visit counter on a Web page, or

a special marker intended to limit or prevent the unauthorized replay or reproduction of a file.

Because a nonce changes with time, it is easy to tell whether or not an attempt at replay or reproduction of a file is legitimate; the current time can be compared with the nonce. If it does not exceed it or if no nonce exists, then the attempt is authorized. Otherwise, the attempt is not authorized.

Nonces are commonly implemented as pseudo-random strings.

2.1.2 Notation

In the following, we give the standard, informal notation with which security protocols are generally described in the literature. We are going to use this notation for the major part of this work, when using an informal language. In particular, it has already been used in the Introduction, when the "Needham-Schroeder public key protocol" has been described and it will be used in Chapter 5, where some secure procedures for the deliveries of digital certificates will be presented. When informally describing secure multicast protocols (Chapter 3 and Appendix A), the notation will slightly change, due to the insertion of labels to indicate some transmitted packets or to indicate the typology of the transmission (*i.e.*, unicast, multicast or broadcast). However, to avoid confusion, the opportune notation will be reminded throughout the work, when needed.

A set of agents able to send and receive messages is here considered. The sending and reception of a message is denoted as $i \quad A \mapsto B : msg$, where msg is the exchanged message and i is the i-th communication channel, over which the exchange takes place. A and B are the sender and the receiver of msg, respectively.

When considering a malicious agent, we will generally denote it as X. X can intercept and also fake messages:

$$\begin{array}{ll} (1) & X(A) \mapsto B : msg \\ (2) & A \mapsto X(B) : msg \end{array}$$

Notation (1) describes X that sends a message msg to B pretending to be A (forgery); (2) denotes: msg, originally intended for B, is actually intercepted by X (interception).

Notation that recurs periodically throughout the thesis is:

$$
\begin{array}{lcl}
name_i, pin_i, etc & := & name\, of\, agent\, i,\ password\, of\, agent\, i,\ etc. \\
pk_i, pk_i^{-1} & := & respectively,\ public\, and\, private\, key\, of\, agent\, i \\
\{...\}_{pk_i^{-1}} & := & message\, signed\, by\, agent\, i \\
\{...\}_{pk_i} & := & message\, encrypted\, by\, public\, key\, of\, agent\, i \\
\{...\}_K & := & message\, encrypted\, by\, symmetric\, key\, K \\
h\{m\} & := & fingerprint\, of\, message\, m \\
mac(m, K) & := & message\, authentication\, code\, of\, message\, m\, with\, key\, K
\end{array}
$$

The reader is invited to note that somewhere, throughout the work, public and private key of agent i will be denoted as $pk(i), sk(i)$ instead of pk_i, pk_i^{-1}.

2.2 Automata

Automata are a model underlying formal specifications of systems. An automaton consists of a set of states, a set of actions, a set of labeled transitions between states, and a set of initial states. Labels represent actions and a transition's label indicates the action causing the transition from one state to another.

A synchronized automaton over a set of automata is an automaton, determined by the way in which its constituting automata cooperate by means of synchronized transitions. The label of a transition is the action being simultaneously executed. When the synchronized automaton changes state by executing an action, all automata which participate simultaneously change state by executing that action, while all others remain idle.

A team automaton, [Ell97, Bee03] is defined in a way similar to the definition of synchronized automata. By starting from a set of component automata, whose actions are divided into input, output and internal actions, a team automaton over this set of components is defined by choosing the synchronizations of actions of its constituting component automata.

2.2.1 Team Automata

A team automaton consists of component automata—ordinary automata without final states and with a distinction of their sets of actions into input, output, and internal actions—combined in a coordinated way such that they can perform shared

actions. Internal actions have strictly local visibility and cannot be used for communication with other component automata, while input and output actions together form the external actions that are observable by other components and that are used for communication between components. During each clock tick the components within a team can simultaneously participate in one instantaneous action, *i.e.*, synchronize on this action, or remain idle. Component automata can thus be combined in a loose or more tight fashion depending on which actions are to be synchronized, and when. Team automata can in turn be used as components in a higher-level team automaton.

Technically, team automata are an extension of I/O automata. However, whereas I/O automata are required to be input enabled, *i.e.*, in each state it must be possible to execute every input action, such a restriction does not hold for component (and team) automata. Moreover, the composition of a set of component automata need not result in a unique team automaton, but rather a whole range of team automata—distinguishable only by their synchronizations—can be composed over a set of component automata. I/O automata, on the other hand, are uniquely defined by their constituents. Finally, I/O automata do not allow output actions to be synchronized, whereas team automata do.

Apart from I/O automata, team automata also possess several features bearing a close resemblance to characteristics of other models from the literature. The distinction of actions into input, output, and internal actions originates from I/O automata and it in fact occurs also in other models based on I/O automata, like the aforementioned interactive state machines and reactive transition systems [CC02]. Similarly, the internal or silent action τ in process algebras like CCS [Mil80] denotes a handshake communication, *i.e.*, the synchronization of two complementary (input and output) actions. Furthermore, the synchronization on common actions also occurs in many other automata-based models in which automata can be composed, like the mixed product of automata [Dub86] and (timed) cooperating (pushdown) automata [DH94, HH94, LMSP00]. The handshake communication in CCS is of a slightly different nature. Many process algebras nevertheless contain specific parallel composition operators that allow processes to communicate by means of synchronizations [BPS01], like CSP [Hoa85].

The main distinction between team automata and many other models from the literature is the freedom they offer by allowing one to *choose* the synchronizations when composing a team over a set of component automata. Most automata-based models, on the contrary, use a single and very strict method of composition, in effect resulting in composite automata that are uniquely defined by their constituents. This holds for all the above mentioned automata-based models, while

this principle also appears in disguise in several non-automata-based models, like CSP and statecharts [Har87].

We now fix some notations and terminology used throughout the thesis, after which we recall some definitions and results concerning team automata from [BEKR03, BK03].

For convenience we denote the set $\{1, \ldots, n\}$ by $[n]$. The (Cartesian) product of sets V_i, with $i \in [n]$, is denoted by $\prod_{i\in[n]} V_i$. In addition to the prefix notation, we also use the infix notation $V_1 \times \cdots \times V_n$. For $j \in [n]$, $\text{proj}_j : \prod_{i\in[n]} V_i \to V_j$ is defined by $\text{proj}_j((a_1, \ldots, a_n)) = a_j$. The powerset of a set V is denoted by 2^V.

Let Σ and Γ be sets of symbols, $\Gamma \subseteq \Sigma$. The morphism $\text{pres}_{\Sigma,\Gamma} : \Sigma^* \to \Gamma^*$, defined by $\text{pres}_{\Sigma,\Gamma}(a) = a$ if $a \in \Gamma$ and $\text{pres}_{\Sigma,\Gamma}(a) = \lambda$ (the empty string) otherwise, preserves the symbols from Γ and erases all other symbols. We discard Σ when no confusion can arise.

Let $f : A \to A'$ and $g : B \to B'$ be functions. Then $f \times g : A \times B \to A' \times B'$ is defined as $(f \times g)(a, b) = (f(a), g(b))$. We use $f^{[2]}$ as shorthand for $f \times f$.

Definition 1 *An* automaton *is a construct $\mathcal{A} = (Q, \Sigma, \delta, I)$, with set Q of* states*, set Σ of* actions*, $Q \cap \Sigma = \varnothing$, set $\delta \subseteq Q \times \Sigma \times Q$ of* transitions*, and set $I \subseteq Q$ of* initial states*.*

The set $\mathbf{C}_\mathcal{A}$ of computations *of $\mathcal{A}$ is defined as consisting of all the sequences $\alpha = q_0 a_1 q_1 \cdots a_n q_n$, where $n \geq 0$ and $q_0 \in I$, and for all $i \in [n]$: $q_i \in Q$, $a_i \in \Sigma$, and $(q_{i-1}, a_i, q_i) \in \delta$.*

The Γ-behaviour $\mathbf{B}^\Gamma_\mathcal{A}$ *of $\mathcal{A}$, with $\Gamma \subseteq \Sigma$, is defined as $\mathbf{B}^\Gamma_\mathcal{A} = pres_\Gamma(\mathbf{C}_\mathcal{A})$.*

The Σ-behaviour of $\mathcal{A}$ is also called the *behaviour* of $\mathcal{A}$, in which case Σ may be discarded from the notation. Let $a \in \Sigma$. The set δ_a of *a-transitions* of $\mathcal{A}$ is defined as $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\}$. Finally, note that behavioral inclusion defines a preorder relation on automata.

As said before, team automata are composed over component automata, which are automata which distinguish *input*, *output*, and *internal* actions.

Definition 2 *A* component automaton *is a construct $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$, with* underlying automaton *$(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ and pairwise disjoint sets Σ_{inp} of* input*, Σ_{out} of* output*, and Σ_{int} of* internal actions*.*

The set Σ denotes the set $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$ of *actions* of the component automaton $\mathcal{C}$ and Σ_{ext} denotes its set $\Sigma_{inp} \cup \Sigma_{out}$ of *external* actions.

For the sequel we let $\mathcal{S} = \{\mathcal{C}_i \mid i \in [n]\}$ be an arbitrary but fixed set of component automata specified as $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$, with set

$\Sigma_i = \Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$ of actions and set $\Sigma_{i,ext} = \Sigma_{i,inp} \cup \Sigma_{i,out}$ of external actions.

When composing team automata over $\mathcal{S}$, the internal actions of the components constituting $\mathcal{S}$ must be private, *i.e.*, uniquely associated to one component automaton. This is formally expressed by requiring that $\Sigma_{i,int} \cap \bigcup_{j \in ([n]-\{i\})} \Sigma_j = \varnothing$, for all $i \in [n]$, *i.e.*, no internal action of any component from $\mathcal{S}$ may appear as an action in any of the other components constituting $\mathcal{S}$. (Recall that, by definition, $\Sigma_{i,int}$, $\Sigma_{i,out}$ and $\Sigma_{i,inp}$ are disjoint.) If this is the case, then $\mathcal{S}$ is called a *composable system* and for the sequel we let $\mathcal{S}$ be a composable system.

The state space of a team automaton composed over $\mathcal{S}$ is the product of the state spaces of the components constituting $\mathcal{S}$. Also the set of actions of a team automaton over $\mathcal{S}$ is uniquely determined. The internal actions of the components are the internal actions of the team. Each action which is output for one or more of the components is an output action of the team. In particular, an action that is an output action of one component and also an input action of another component, is considered an output action of the team. The input actions of the team that do not occur at all as an output action of any of the components constituting $\mathcal{S}$, are the input actions of the team. The reason for this construction is as follows. When relating an input action a of a component to an output action a of another component, the input may be thought of as being caused by the output. On the other hand, the output action remains observable as output to other components. Finally, the transitions of a team over $\mathcal{S}$ are based on but not fixed by those of the components constituting $\mathcal{S}$. They are chosen by allowing certain *synchronizations* on actions, while excluding others.

Definition 3 *Let* $a \in \bigcup_{i \in [n]} \Sigma_i$. *The set* $\Delta_a(\mathcal{S})$ *of* synchronizations *of* a *is defined as* $\Delta_a(\mathcal{S}) = \{(q, q') \in \prod_{i \in [n]} Q_i \times \prod_{i \in [n]} Q_i \mid [\exists\, j \in [n] : \mathit{proj}_j{}^{[2]}(q, q') \in \delta_{j,a}] \wedge [\forall\, i \in [n] : [\mathit{proj}_i{}^{[2]}(q, q') \in \delta_{i,a}] \vee [\mathit{proj}_i(q) = \mathit{proj}_i(q')]]\}$.

The set $\Delta_a(\mathcal{S})$ thus contains all possible combinations of a-transitions of the components constituting $\mathcal{S}$, with all non-participating components remaining idle. It is explicitly required that in every synchronization at least one component participates. The state change of a team automaton over $\mathcal{S}$ is thus defined by the local state changes of the components constituting $\mathcal{S}$ that participate in the action of the team being executed. Hence, when defining a team automaton over $\mathcal{S}$, a specific subset of $\Delta_a(\mathcal{S})$ must be chosen for each action a. This enforces a certain kind of communication between the components constituting the team.

Definition 4 *A* team automaton *over the* composable system $\mathcal{S}$ *is a construct* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$*, with* $Q = \prod_{i\in[n]} Q_i$*,* $\Sigma_{inp} = (\bigcup_{i\in[n]} \Sigma_{i,inp}) - \Sigma_{out}$*,* $\Sigma_{out} = \bigcup_{i\in[n]} \Sigma_{i,out}$*,* $\Sigma_{int} = \bigcup_{i\in[n]} \Sigma_{i,int}$*,* $\delta \subseteq Q \times \Sigma \times Q$*, where* $\Sigma = \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$*, is such that* $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\} \subseteq \Delta_a(\mathcal{S})$*, for all* $a \in \Sigma$*, and* $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\} = \Delta_a(\mathcal{S})$*, for all* $a \in \Sigma_{int}$*, and* $I = \prod_{i\in[n]} I_i$*.*

Each choice of synchronizations thus defines a team automaton. It is important to observe that *every team automaton is again a component automaton*, which in its turn can be used as a component in an *iteratively* composed team. In this way one can construct, *e.g.*, a team automaton $\mathcal{T}'$ over the composable system $\{\mathcal{T}'', \mathcal{C}_3\}$, where $\mathcal{T}''$ is a team composed over the composable system $\{\mathcal{C}_1, \mathcal{C}_2\}$.

Figure 2.1 describes the transition space $\Delta_a(\mathcal{S})$ of a team automaton over S. The choices of team transition relations $\delta_a, \forall a \in \Sigma$, define a specific TA.

It may be useful, though, to *hide* certain external actions of a team automaton before using this team in an iterative composition, in order to prohibit synchronizations on these actions on a higher level of the composition.

Definition 5 *Let* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *be a team automaton and let* $\Gamma \subseteq \Sigma_{ext}$*. Then hide*$_\Gamma(\mathcal{T}) = (Q, (\Sigma_{inp} - \Gamma, \Sigma_{out} - \Gamma, \Sigma_{int} \cup \Gamma), \delta, I)$*.*

In hide$_\Gamma(\mathcal{T})$, the external actions in Γ have thus become unobservable for other automata by regarding them as internal actions. Without formally defining renaming, we assume these actions to be indexed in order to guarantee composability.

It may sometimes be useful to construct unique team automata of a specified type. In [BEKR03] several fixed strategies for choosing the synchronizations of a team automaton were defined, each leading to a uniquely defined team automaton. These strategies fix the synchronizations of a team by defining, per action a, certain conditions on the a-transitions to be chosen from $\Delta_a(\mathcal{S})$, thus determining a unique subset of $\Delta_a(\mathcal{S})$ as the set of a-transitions of the team. Once such subsets have been chosen for all actions, the team automaton over $\mathcal{S}$ that it defines is unique.

Definition 6 *Let* $a \in \bigcup_{i\in[n]} \Sigma_i$*. The set* is-ai *for* a *in* $\mathcal{S}$*, denoted by* $\mathcal{R}_a^{ai}(\mathcal{S})$*, is defined as* $\mathcal{R}_a^{ai}(\mathcal{S}) = \{(q, q') \in \Delta_a(\mathcal{S}) \mid \forall\, i \in [n] : [a \in \Sigma_i \Rightarrow proj_i^{[2]}(q, q') \in \delta_{i,a}]\}$*.*

The set $\mathcal{R}_a^{ai}(\mathcal{S})$ thus contains *all and only* those a-transitions from $\Delta_a(\mathcal{S})$ in which every component automaton with a as an action participates. Hence the team

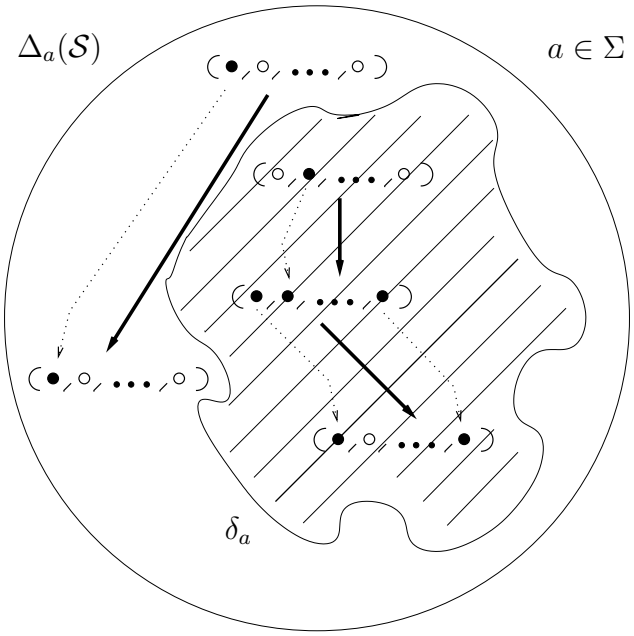

Figure 2.1: Transition space of TA

automaton over $\mathcal{S}$ defined by this set is the unique team automaton in which any execution of a sees the participation of all components having a in their set of actions.

Definition 7 The max-ai team automaton *over* $\mathcal{S}$*, denoted by* $|||\,\mathcal{S}$*, is* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *if* $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$*, for all* $a \in \Sigma$*.*

Informally, the *max-ai team automaton* over $\mathcal{S}$ is the TA in which the synchronization is defined on all, and only, those transitions in which, for each action, all the component automata featuring that action participate to the transition. ai stands for *action indispensable*.

Figure 2.2 shows two component automata $\mathcal{C}_1$ and $\mathcal{C}_2$. Figure 2.3 shows two of the several team automata that can be built by starting from those component automata, in particular, by choosing their transitions. We enforce maximal synchronization in $\mathcal{T}^{ai} = |||\,\{\mathcal{C}_1, \mathcal{C}_2\}$, that is the resulting team where any execution of action a and action b sees the participation of all the two components (that, of course, have both a and b in their set of actions), Definition 7. $\mathcal{T}^{free}$ has not been

formally defined. Informally, it is the team automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$ where any execution of action a and action b sees the participation of only one component. In both the figures, an informal description of TA by their interactions is given.

The Γ-*behaviour* of a team automaton $\mathcal{T}$, denoted as $\mathbf{B}^{\Gamma}_{\mathcal{T}}$, is defined as usual in automata theory (see Definition 1). In particular, $\mathbf{B}^{\Gamma}_{\mathcal{T}} = \text{pres}_{\Gamma}(\mathbf{C}_{\mathcal{T}})$, with set $\mathbf{C}_{\mathcal{T}}$ of computations of $\mathcal{T}$ consisting of all the sequences $\alpha = q_0 a_1 q_1 \ldots a_n q_n$, where $n \geq 0$ and q_0 is an initial state, q_i, are states, a_i are actions and (q_{i-1}, a_i, q_i) are transitions.

Along with this general notion of behaviour, other notions can be defined. When $\Gamma = \Sigma_{out}$, then $\mathbf{B}^{\Sigma_{out}}_{\mathcal{T}}$ is the output behaviour of $\mathcal{T}$. By opportunely choosing Γ, also the input and the internal behaviour of $\mathcal{T}$ can be defined.

Remark 1 *In [BEKR03] it was shown that the behaviour of an iteratively composed max-ai team automaton equals that of the max-ai team automaton over the underlying components,* i.e.*, continuing our above example: if $\mathcal{T}'$ and $\mathcal{T}''$ are the max-ai team automata over $\{\mathcal{T}'', \mathcal{C}_3\}$ and $\{\mathcal{C}_1, \mathcal{C}_2\}$, respectively, and $\mathcal{T}$ is the max-ai team automaton over $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$, then $\mathbf{B}_{\mathcal{T}'} = \mathbf{B}_{\mathcal{T}}$.*

A team automaton is said to satisfy *compositionality* if its behaviour can be described in terms of that of its constituting component automata, *i.e.*, when the sequences forming the behaviour of a set of component automata can be *shuffled* in such a way that the sequences forming the behaviour of a particular team over these components result.

Definition 8 *Let Δ_i be alphabets. The* full synchronized shuffle $\underline{||}_{\{\Delta_i | i \in [n]\}} L_i$ *of $L_i \subseteq \Delta_i^*$, with $i \in [n]$, is defined as $\underline{||}_{\{\Delta_i | i \in [n]\}} L_i = \{w \in (\bigcup_{i \in [n]} \Delta_i)^* \mid \forall\, i \in [n] : \textit{pres}_{\Delta_i}(w) \in L_i\}$.*

Example 1 *Let Δ_1, Δ_2 be alphabets. Let $L_1 = \{abc\}$ be a sequence such that $L_1 \subseteq \Delta_1 = \{a, b, c\}$ and $L_2 = \{cd\}$ a second sequence such that $L_2 \subseteq \Delta_2 = \{c, d\}$. Then, the full synchronized shuffle $abc \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; cd = \{abcd\}$ (*i.e.*, words must synchronize on $\Delta_1 \cap \Delta_2 = \{c\}$).*

In [BK03] it was shown that the construction of team automata according to certain types of synchronization, like the one leading to max-ai team automata, guarantees compositionality.

Theorem 1 (Compositionality of team automata) *Let $\mathcal{T}$ be the max-ai team automaton over $\mathcal{S}$. Then $\mathbf{B}_{\mathcal{T}} = \underline{||}_{\{\Sigma_i | i \in [n]\}} \mathbf{B}_{\mathcal{C}_i}$.*

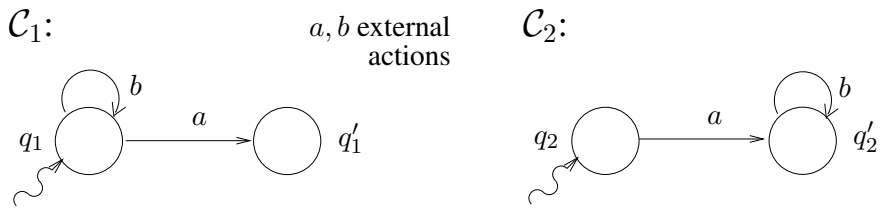

Figure 2.2: Example CA

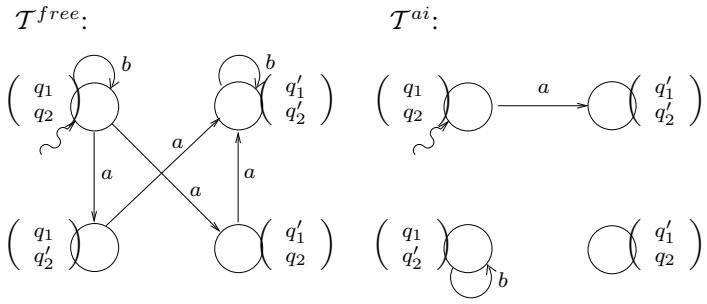

Figure 2.3: Example TA

2.3 Process algebras

Process algebras are executable specification languages for the description of concurrent and distributed systems. Systems are composed of agents (also called processes). Each process follows a certain behaviour, represented by a set of atomic actions. A process can perform these actions both independently or by interacting with each other. All the single actions are tied together through a set of operators (*e.g.*, , a non deterministic choice, or a sequentialization, or a parallel composition) and the overall result forms the specification for that system.

To facilitate a comparison between processes, several notions of behavioral equivalences have been defined in the literature. To our aim, we mainly deal with the notion of *weak bisimulation*, introduced in the following.

Classical process algebras (among the others CCS (the Calculus of Communicating Systems, [Mil89]) and CSP ([RSG+00], Communicating Sequential Processes) have been successively extended to cope with the possibility to detect flows in security protocols, *e.g.*, [LR97, FG97b, Mar03] and even by expressing features like mobility, *e.g.*, [AG97, NNHJ99, AFG98] and time, *e.g.*,

[HR95, GLM03, BR04].

2.3.1 Crypto-CCS and tCryptoSPA

Here, two formal process algebras for the modeling of cryptographic protocols are presented.

Crypto-CCS

This subsection presents a concise description of the Crypto-CCS syntax and semantics. Some constructs of the language are here omitted, since they are not of direct interest for the investigated topics. For a complete description, the interested reader is invited to see [Mar03].

The model of the language consists of sequential agents able to communicate by exchanging messages.

The data handling part of the language consists of messages and inference systems. Messages are the data manipulated by agents, they form a set *Msgs* of terms possibly containing variables. The set *Msgs* is defined by the grammar:

$$m ::= x \mid b \mid F^1(m_1, \ldots, m_{k_1}) \mid \ldots \mid F^l(m_1, \ldots, m_{k_l})$$

where F^i (for $1 \leq i \leq l$) are the constructors for messages, $x \in V$ is a countable set of variables, $b \in B$ is a collection of basic messages and k_i, for $1 \leq i \leq l$, gives the number of arguments of the constructor F^i. Messages without variables are *closed* messages.

Inference systems model the possible operations on messages. They consist of a set of rules r, *e.g.*, :

$$r = \frac{m_1 \quad \ldots \quad m_n}{m_0}$$

where $\{m_1, \ldots, m_n\}$ is a set of premises (possibly empty) and m_0 is the conclusion. An instance of the application of rule r to closed messages m_i is denoted as $m_1 \quad \ldots \quad m_n \vdash_r m_0$. Given an inference system, a *deduction function* $\mathcal{D}$ is defined such that, if ϕ is a finite set of closed messages, then $\mathcal{D}(\phi)$ is the set of closed messages that can be deduced starting from ϕ by applying instances of the rules in the system. The syntax and semantics of Crypto-CCS are parametric with respect to a given inference system. Example inference systems suitable to model specific cryptographic protocols will be shown in the following sections.

The control part of the language consists of compound systems, *i.e.*, sequential agents running in parallel. The language syntax is as follows:

COMPOUND SYSTEMS:	$S ::= (S_1 \| \| S_2) \mid S \setminus C \mid A_\phi$
SEQUENTIAL AGENTS:	$A ::= \mathbf{0} \mid p.A \mid A_1 + A_2 \mid [m_1 \ldots m_n \vdash_r x]A_1; A_2$
	$\mid [m = m']A_1; A_2 \mid E(m_1, \ldots, m_n)$
PREFIX CONSTRUCTS:	$p ::= c!m \mid c?x$

where $m, m', m_1, \ldots, m_n$ are *closed* messages or variables, x is a variable, $c \in Ch$ (a finite set of channels) ϕ is a finite set of *closed* messages, C is a subset of Ch.

$\mathbf{0}$ is the process that does nothing.

$p.A$ is the process that can perform an action according to the particular prefix construct p and then behaves as A. In particular,

- $c!m$ denotes a message m sent on channel c;
- $c?x$ denotes the receiving of a message m on channel c. The received message replaces the variable x.

$A_1 + A_2$ represents the non deterministic choice between A and A_1.

$[m_1 \ldots m_n \vdash_r x]A_1; A_2$ is the inference construct. If, by applying an instance of rule r, with premises $m_1 \ldots m_n$, a message m can be inferred, then the process behaves as A_1 (where m replaces x), otherwise it behaves as A_2.

$[m = m']A_1; A_2$ is the match construct, to check message equality. If $m = m'$ then the system behaves as A_1, otherwise it behaves as A_2.

A compound system $S_1||S_2$ denotes the parallel execution of S_1 and S_2. $S_1||S_2$ performs an action p if one of its sub-components performs p. A synchronization, or internal action, denoted by τ, may take place whenever S_1 and S_2 are able to perform two complementary actions, *i.e.*, send-receive actions on the same channel.

A compound system $S \setminus C$ allows only visible actions whose channels are not in C. (Internal action τ being the invisible action).

The term A_ϕ is a single sequential agent whose knowledge, *i.e.*, the set of messages which occur in its term, is described by ϕ. The knowledge of an agent increases either when it receives messages (see rule (?) in Fig. 2.4) or it infers new messages from the messages it knows (see rule $\mathcal{D}$ in Fig. 2.4). For every sequential agent A_ϕ, it is required that all the closed messages that appear in A_ϕ belong to its knowledge ϕ.

The activities of the agents are described by the actions that they can perform. The set Act of actions which may be performed by a compound system ranges over by a and it is defined as: $Act = \{c?m, c!m, \tau \mid c \in C, m \in Msgs, m \text{ closed}\}$. $\mathcal{P}$ is the set of all the Crypto-CCS *closed* terms (*i.e.*, with no

$$(!)\frac{}{(c!m.A)_\phi \xrightarrow{c!m} (A)_\phi} \qquad (||_1)\frac{S \xrightarrow{a} S'}{S||S_1 \xrightarrow{a} S'||S_1}$$

$$(||_2)\frac{S \xrightarrow{c!m} S' \quad S_1 \xrightarrow{c?m} S_1'}{S||S_1 \xrightarrow{\tau} S'||S_1'}$$

$$(?)\frac{m \in Msgs}{(c?x.A)_\phi \xrightarrow{c?m} (A[m/x])_{\phi\cup\{m\}}} \qquad (\backslash_1)\frac{S \xrightarrow{c!m} S' \quad c \notin L}{S \setminus L \xrightarrow{c!m} S' \setminus L}$$

$$(\mathcal{D})\frac{m_1 \ldots m_n \vdash_r m \quad (A[m/x])_{\phi\cup\{m\}} \xrightarrow{a} (A')_{\phi'}}{([m_1 \ldots m_n \vdash_r x]A; A_1)_\phi \xrightarrow{a} (A')_{\phi'}} \qquad (+_2)\frac{S \xrightarrow{a} S'}{S + S_1 \xrightarrow{a} S'}$$

$$(\mathcal{D}_1)\frac{\nexists m \text{ s.t. } m_1 \ldots m_n \vdash_r m \quad (A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}{([m_1 \ldots m_n \vdash_r x]A; A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}$$

$$(=)\frac{m = m' \quad (A)_\phi \xrightarrow{a} (A')_{\phi'}}{([m = m']A; A_1)_\phi \xrightarrow{a} (A')_{\phi'}}$$

$$(=_1)\frac{m \neq m' \quad (A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}{([m = m']A; A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}$$

$$(Const)\frac{E(x_1, \ldots, x_n) =_{def} A \quad A[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} A_1}{E(m_1, \ldots, m_n) \xrightarrow{a} A_1}$$

Figure 2.4: Operational semantics of Crypto-CCS.

free variables). *sort(P)* is the set of all the channels that syntactically occur in the term *P*.

The operational semantics of a Crypto-CCS term is described by means of the *labeled transition system* (*lts*, for short) $\langle \mathcal{P}, Act, \{\xrightarrow{a}\}_{a\in Act}\rangle$, where $\{\xrightarrow{a}\}_{a\in Act}$ is the least relation between Crypto-CCS processes induced by the axioms and inference rules of Fig. 2.4 (in that figure the symmetric rules for $||_1, ||_2, \backslash_1, +_2$ are omitted).

The expression $S \xrightarrow{a} S'$ means that the system can move from the state S to the state S' through the action a. The expression $S \Longrightarrow S'$ denotes that S and S' belong to the reflexive and transitive closure of $\xrightarrow{\tau}$; let $\gamma = a_1 \ldots a_n \in (Act\backslash\{\tau\})^*$ be a sequence of actions. Then, $S \overset{\gamma}{\Longrightarrow} S'$ if $S \Longrightarrow \xrightarrow{a_1} \Longrightarrow \ldots \Longrightarrow \xrightarrow{a_n} \Longrightarrow S'$.

As behavioral relations among Crypto-CCS terms, we are interested in trace inclusion (equivalence) and (weak) simulation.

Definition 9 *We say that the traces of P are included in the traces of Q ($P \leq_{trace} Q$) whenever, if $P \stackrel{\gamma}{\Longrightarrow} P_1$ then $Q \stackrel{\gamma}{\Longrightarrow} Q_1$. We write that $P =_{trace} Q$ iff $P \leq_{trace} Q$ and $Q \leq_{trace} P$.*

Definition 10 *We say that a relation $\mathcal{R}$ among processes is a weak simulation, if for every $(P, Q) \in \mathcal{R}$ we have:*

- *If $P \stackrel{a}{\longrightarrow} P', a \neq \tau$, then there exists Q' s.t. $Q \stackrel{a}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$.*
- *If $P \stackrel{\tau}{\longrightarrow} P'$ then there exists Q' s.t. $Q \Longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$.*

The union of all weak simulations is a weak simulation and it is denoted by $\prec$. As usual, it holds that if $P \prec Q$ then $P \leq_{trace} Q$.

tCryptoSPA

The real-time extension of the *Cryptographic Security Process Algebra* (for short, $CryptoSPA$) of [FM99, FGM00a] has been proposed in [GLM03]. The new language, $timedCryptoSPA$ ($tCryptoSPA$ for short), is adopted for describing cryptographic protocols where information about the concrete timing of events is necessary. We remind the reader of the syntax, the operational semantics of the language and some auxiliary notions. The description is not exhaustive, since some constructs are not of direct interest for the investigated topics. Furthermore, some terms of the language are the same as in the Crypto-CCS language (previous subsection, 2.3.1). Finally, the interested reader is referred to [GLM03] for a more complete discussion of $tCryptoSPA$.

The set $\mathcal{L}$ of $tCryptoSPA$ processes is defined as:

$$P ::= \mathbf{0} \mid c(x).P \mid \overline{c}m.P \mid \tau.P \mid tick.P \mid P_1 + P_2 \mid P_1 || P_2 \mid P \backslash L \mid$$

$$A(m_1, \ldots, m_n) \mid [\langle m_1, \ldots, m_r \rangle \vdash_{rule} x] P_1; P_2$$

We omit to describe terms whose meaning has been already explained in Subsection 2.3.1. To this aim, note that the $tCryptoSPA$ sequential construct $\overline{c}e.P$ is syntactically and semantically equivalent to the Crypto-CCS sequential construct $c!m.P$. Thus, $\overline{c}m.P$ is the process that can send m on channel c, then behaving like P.

$m, m_1, \ldots, m_r, m_n$ are messages or variables and L is a set of channels. Both the operators $c(x).P$ and $[\langle m_1 \ldots m_r \rangle \vdash_{rule} x]P_1; P_2$ bind the variable x in P and P_1, respectively.

Let $Def : Const \longrightarrow \mathcal{L}$ be a set of defining equations of the form $A(x_1, \ldots, x_n) \doteq P$, where P may contain no free variables except $x_1, \ldots, x_n$, which must be distinct. Constants permit us to define recursive processes. A term P is *closed* with respect to Def if all the constants occurring in P are defined in Def (and, recursively, for their defining terms). A term P is *guarded* w.r.t. Def if all the constants occurring in P (and, recursively, for their defining terms) occur in a prefix context [Mil89].

The set Act of actions which may be performed by a system is defined as: $Act = \{c(m), \overline{c}m, \tau, tick, \mid c \in \mathcal{I}, \overline{c} \in \mathcal{O}, m \in \mathcal{M}, m \text{ closed}\}$. τ is the internal, invisible action. $tick$ is the special action used to model time elapsing. We let l range over $Act \backslash \{tick\}$. We call $\mathcal{L}$ the set of all the ***tCryptoSPA closed*** terms (***i.e.***, with no free variables) that are closed and guarded w.r.t. Def. We define ***sort(P)*** to be the set of all the channels syntactically occurring in the term P.

$\tau.P$ is the process that executes the internal, invisible action τ and then behaves like P;

$tick.P$ is a process willing to let one time unit pass and then behaving as P;

$P_1 + P_2$ (***choice***) represents the nondeterministic choice between the two processes P_1 and P_2; with respect to $tick$ actions, time passes when both P_1 and P_2 are able to perform a *tick* action – and in such a case by performing $tick$ a configuration where both the derivatives of the summands can still be chosen is reached. When only one of the two processes can perform $tick$, say P_1, it could be either that P_1 performs $tick$ – and in such a case P_2 is discarded – or P_2 performs its normal activity – and in such a case P_1 is discarded; moreover, τ prefixed summands have priority over $tick$ prefixed summands;

$P_1 || P_2$ (***parallel***) is the parallel composition of processes that can proceed in an asynchronous way but they must synchronize on complementary actions to make a communication, represented by a τ. Both components must agree on performing a *tick* action, and this can be done even if a communication is possible.

$P \backslash L$ allows only visible actions whose channels are not in L;

$A(m_1, \ldots, m_n)$ behaves like the respective defining term P where all the variables $x_1, \ldots, x_n$ are replaced by the messages $m_1, \ldots, m_n$.

The time model adopted in the language is known as the ***fictitious clock*** approach of, ***e.g.***, [HR95]. A global clock is supposed to be updated whenever all the processes agree on this, by globally synchronizing on the special action $tick$,

representing the passing of a time unit. All the other actions are assumed to take no time.

It holds that also for $tCryptoSPA$ the syntax and the semantics are parametric with respect to the given inference system.

The operational semantics of a $tCryptoSPA$ term is described by means of the *labeled transition system* (*lts*, for short) $\langle \mathcal{L}, Act, \{\stackrel{a}{\longrightarrow}\}_{a\in Act}\rangle$, where $\{\stackrel{a}{\longrightarrow}\}_{a\in Act}$ is the least relation between $tCryptoSPA$ processes induced by the axioms and inference rules of Figure 2.5 (symmetric rules for $+_1, +_3, ||_1, ||_2$ and $\backslash L$ are omitted).

The expression $P \stackrel{a}{\Rightarrow} P'$ is a shorthand for $P(\stackrel{\tau}{\longrightarrow})^* P_1 \stackrel{a}{\longrightarrow} P_2(\stackrel{\tau}{\longrightarrow})^* P', a \neq \tau$, where $(\stackrel{\tau}{\longrightarrow})^*$ denotes a (possibly empty) sequence of transitions labeled τ. The expression $P \Rightarrow P'$ is a shorthand for $P(\stackrel{\tau}{\longrightarrow})^* P'$. Let $\gamma = a_1, \ldots, a_n \in (Act\backslash\{\tau\})^*$ be a sequence of actions; then $P \stackrel{\gamma}{\Rightarrow} P'$ iff there exist $P_1, \ldots, P_{n-1} \in \mathcal{P}$ such that $P \stackrel{a_1}{\Rightarrow} P_1 \stackrel{a_2}{\Rightarrow}, \ldots, P_{n-1} \stackrel{a_n}{\Rightarrow} P'$. Let $\mathbf{0}' \doteq tick.\mathbf{0}'$.

For timed behavioural relations among $tCryptoSPA$ processes, we will be mainly interested in *timed trace* inclusions.

Definition 11 *For any $P \in \mathcal{L}$ the set $T(P)$ of timed traces associated with P is defined as follows $T(P) = \{\gamma \in (Act\backslash\{\tau\})^* \mid \exists P'.P \stackrel{\gamma}{\Rightarrow} P'\}$. The timed trace pre-order, denoted by $\leq_{ttrace}$, is defined as follows: $P \leq_{ttrace} Q$* iff *$T(P) \subseteq T(Q)$. P and Q are* timed trace equivalent*, denoted by $P =_{ttrace} Q$, if $T(P) = T(Q)$.*

We define the concept of weak simulation as usual.

Definition 12 *We say that a relation R among processes is a weak simulation, if for every $(P, Q) \in \mathcal{R}$ we have:*

- *If $P \stackrel{a}{\longrightarrow} P', a \neq \tau$, then there exists Q' s.t. $Q \stackrel{a}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$.*
- *If $P \stackrel{\tau}{\longrightarrow} P'$ then there exists Q' s.t. $Q \Longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$.*

Let $\prec$ the union of all weak simulations among processes. Then, we have $\prec \subseteq \leq_{ttrace}$.

2.4 GNDC and tGNDC

In this section, we present the general schema *Generalized Non Deducibility on Compositions* (GNDC), for the definition of security properties given in [FM99],

$$(input)\frac{m \in \mathcal{M}}{c(x).P \xrightarrow{c(m)} P[m/x]} \qquad (output)\frac{}{\overline{c}m.P \xrightarrow{\overline{c}m} P} \qquad (internal)\frac{}{\tau.P \xrightarrow{\tau} P}$$

$$(tick)\frac{}{tick.P \xrightarrow{tick} P} \qquad (||_1)\frac{P_1 \xrightarrow{l} P_1'}{P_1||P_2 \xrightarrow{l} P_1'||P_2} \qquad (||_2)\frac{P_1 \xrightarrow{c(x)} P_1' \quad P_2 \xrightarrow{\overline{c}m} P_2'}{P_1||P_2 \xrightarrow{\tau} P_1'||P_2'}$$

$$(||_3)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \xrightarrow{tick} P_2'}{P_1||P_2 \xrightarrow{tick} P_1'||P_2'} \qquad (\backslash L)\frac{P \xrightarrow{c(m)} P' \quad c \notin L}{P\backslash L \xrightarrow{c(m)} P'\backslash L} \qquad (+_1)\frac{P_1 \xrightarrow{l} P_1'}{P_1 + P_2 \xrightarrow{l} P_1'}$$

$$(+_2)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \xrightarrow{tick} P_2'}{P_1 + P_2 \xrightarrow{tick} P_1' + P_2'} \qquad (+_3)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \not\xrightarrow{tick} \quad P_2 \not\xrightarrow{\tau}}{P_1 + P_2 \xrightarrow{tick} P_1'}$$

$$(Def)\frac{P[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} P' \quad A(x_1, \ldots, x_n) \doteq P}{A(m_1, \ldots, m_n) \xrightarrow{a} P'}$$

$$(\mathcal{D})\frac{\langle m_1, \ldots, m_r\rangle \vdash_{rule} m \quad P_1[m/x] \xrightarrow{a} P_1'}{[\langle m_1, \ldots, m_r\rangle \vdash_{rule} x]P_1; P_2 \xrightarrow{a} P_1'}$$

$$(\mathcal{D})\frac{\nexists m \text{ s.t. } \langle m_1, \ldots, m_r\rangle \vdash_{rule} m \quad P_2 \xrightarrow{a} P_2'}{[\langle m_1, \ldots, m_r\rangle \vdash_{rule} x]P_1; P_2 \xrightarrow{a} P_2'}$$

Figure 2.5: Operational semantics of $tCryptoSPA$.

and its timed extension tGNDC given in [GLM03]. All the specifications that will be referred to are intended to be written by exploiting some variants of the process algebra CCS [Mil89]. In particular, the considered formal language will be Crypto-CCS [FGM04, Mar03], Subsection 2.3.1, when the protocol specification does not involve temporal issues, whereas the $tCryptoSPA$ language will be used to deal with timed properties of a protocol.

The reader is invited to note that in Chapter 4, where a dissertation on the applicability of Team Automata [Bee03] to security will be given, a re-formulation of GNDC in terms of Team Automata will be presented.

In the literature, several efforts have been made to prevent the unauthorized information flow in multilevel computer systems [BP76], *i.e.* systems where processes and objects are bound to a specific security level. An example from military jargon is the fact that documents are generally hierarchized from unclassified to top secret. The seminal idea of *non interference* proposed in [GM82] aims at assuring that information can only flow from low levels to higher ones. The

first taxonomy of non-interference-like properties has been uniformly defined and compared in [FG94, FG97a, FG01] in the context of a CCS-like process algebra. In particular, processes in the algebra were divided into high and low processes, according to the level of actions that they can perform. To detect whether an incorrect information flow (*i.e.* from high to low) has occurred, a particular non-interference-like property has been defined, the so-called *Non Deducibility on Compositions* (NDC). NDC essentially says that a process is secure with respect to wrong information flows if its low behaviour in isolation appears to be the same as its low behaviour when interacting with any high-level process. NDC can be reformulated from the world of multilevel systems to the one of network security. See [FM99, FGM00b], where the low-level process becomes a specification of a cryptographic communication protocol and the behaviour of the protocol running in isolation is compared with that of the protocol running in parallel with any possible adversary.

As a further step, a *Generalized* NDC (GNDC) has been formulated in [FM99], in order to encompass in a uniform way many security properties. The main idea of GNDC is the following: a system P satisfies property $GNDC^{\alpha}_{\lhd}$ if the behavior of P, despite the presence of a hostile environment X that can interact with P only through a fixed set of channels C, *appears* to be same (w.r.t. a behavioral relation $\lhd$ of observational equivalence) to the behavior of a modified version $\alpha(P)$ of P that represents the *expected* (correct) behavior of P. By varying $\alpha(P)$, several security properties can be defined and analyzed within this generalized schema.

The analysis of cryptographic protocols involves specifying a set of messages known by the adversary at the beginning of the computation. This *static* (initial) knowledge of the hostile environment must be bound to a specific set of messages. This limitation is needed to avoid a too strong hostile environment that would be able to corrupt any secret (as it would know all cryptographic keys, etc.). Given an adversary X, we call $ID(X)$ the set of closed messages that syntactically appear in X. This set, intuitively, contains all the messages that are initially known by X. Let ϕ_X be a set of messages representing the static, initial knowledge that we would like to give to X. We want $ID(X)$ to be consistent with ϕ_X. This can be obtained by requiring that all the messages in $ID(X)$ are *deducible* from ϕ_X by means of the *deduction function* $\mathcal{D}$.

The set $\mathcal{E}^{\phi_X}_C$ of processes that can communicate on a subset of public channels C and have an initial knowledge bound by ϕ_X can be therefore defined as follows:

$$\mathcal{E}^{\phi_X}_C = \{X \in \mathcal{P} \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \mathcal{D}(\phi_X)\}$$

We consider as hostile processes only the ones belonging to $\mathcal{E}^{\phi_X}_C$.

We define the property $GNDC^{\alpha}_{\lhd}$ as follows:

Definition 13 *A process P is $GNDC^{\alpha}_{\lhd}$ iff $\forall X \in \mathcal{E}^{\phi_X}_C : (P||X) \setminus C \lhd \alpha(P)$ where $\lhd : \mathcal{P} \to \mathcal{P}$ is a behavioral relation between processes and $\alpha : \mathcal{P} \to \mathcal{P}$ is a function between processes.*

For the sake of completeness, it is worth noticing that a slightly extended GNDC schema has been recently defined in [FGM04], incorporating the fact that the set of bad behaviours of P may depend on P itself and on the property under scrutiny.

For the analysis of safety properties it is enough to consider the trace inclusion relation $\leq_{trace}$ as behavioral relation among the terms of the algebra. When the $\leq_{trace}$ relation is considered, there exists a sufficient criterion for the static characterization, *i.e.*, not involving the universal predicate $\forall$, of $GNDC^{\alpha}_{\lhd}$ properties. In the following, we give hints to the definition of GNDC without the need of the universal predicate, since some notions will be useful in the rest of the chapter. For further details about this static characterization, the interested reader can see [FGM00a, FM99], where the following statements were first declared and proved.

Informally, the so called most powerful intruder in the trace setting ($(Top^C_{trace})_\phi$, hereafter, for short Top^C_ϕ) is that intruder whose knowledge is ϕ, that can communicate only over channels in C, that can receive every message passing over these channels (increasing in such a way its knowledge) and, finally, that can send over these channels every message that it can deduce starting from ϕ.

More formally, Top^C_ϕ is defined as follows in [FM99]:

Definition 14

$$Top^C_\phi = \sum_{c \in C} c(x).Top^C_{\phi \cup \{x\}} + \sum_{c \in C, m \in \mathcal{D}(\phi)} \overline{c}m.Top^C_\phi$$

It has been proved ([FGM00a, FM99]) that the general way in which Top^ϕ_C is specified implies that its behaviour includes that of any X belonging to the set $\mathcal{E}^{\phi_X}_C$ of admissible hostile processes.

Corollary 1 *For every function $\alpha : \mathcal{P} \to \mathcal{P}$, a process P is $GNDC^{\alpha}_{\leq_{trace}}$ iff $(P||Top^C_\phi) \setminus C \leq_{trace} \alpha(P)$.*

The corollary implies that, for the analysis of safety properties in the trace setting, to check if a specification enjoys GNDC w.r.t. all the admissible hostile environments, it is sufficient to check if the same specification enjoys GNDC with respect to the most powerful intruder Top_C^{ϕ}.

By varying the parameter α, the GNDC schema can be used to define and verify many security properties—among which secrecy, integrity, and entity authentication [FG95, FGM00a, FGM00b, FM99, GMPV03a, MPV03]. As an example, we remind here how the secrecy and the entity authentication properties have been formalized in [FGM00a] (relation $\lhd$ for specifying these properties is trace inclusion $\leq_{trace}$).

The requirements for a secrecy property to be satisfied are quite intuitive: a certain message *M*, declared to be secret, should not be learnt by unauthorized users. Thus, let us consider the event *learnt(M)*, signaling that *M* has been learnt by the hostile environment. Then, $\alpha_S(P(m))$ "is the set of processes where the event *learnt(M)* can never occur". For more details, the interested reader can see [FGM00b].

On the other hand, entity authentication "should allow the verification of an entity's claimed identity, by another entity" [FGM00a]. To formalize this action, the followed approach is the one proposed in [Low96] and based on a so called *correspondence* between actions. Let us consider two users *A* and *B*, participating through a protocol. To assure the property, one would like that, whenever *A* concludes the protocol apparently with *B*, *B* has indeed executed the protocol. This can be tested with the introduction of two events, *commit(A,B)* and *run(B,A)*, representing tha fact that A has indeed terminated the protocol apparently with *B*, (action *commit*), and *B* has indeed started communicating with *A*, (action *run*). To fulfill entity authentication means to require that event *commit(A,B)* is always preceded by event *run(B,A)*. In the GNDC definition, $\alpha_{EA}(P)$ is the process where *commit(A,B)* is always preceded by *run(B,A)*.

Along with GNDC, a general schema for the definition of timed security properties, called *timed Generalized Non Deducibility on Compositions* ($tGNDC$ for short) has been proposed in [GLM03].

Property $tGNDC$ rephrases the analogue GNDC, but in a timed setting. A system S is $tGNDC_{\lhd}^{\alpha}$ *iff* for every enemy X the composition of the system with X satisfies the timed specification $\alpha(S)$. Basically, $tGNDC$ guarantees that the timed property α is satisfied, with respect to the $\lhd$ timed behavioural relation, even when the system is composed with any possible adversary X.

We give here the set of admissible hostile environments for our timed setting. For a certain enemy X, we call $ID(X)$ the set of closed messages that syntactically appears in X, all the messages initially known by X. Let ϕ_0 be the initial knowledge we would like to give to the enemy at the beginning of the computation. We require that all the messages in $ID(X)$ are deducible from ϕ_0. We consider as hostile processes only the ones belonging to the set $t\mathcal{E}_C^{\phi_0}$ [1]. They can communicate on a subset of public channels C and have an initial knowledge bound by ϕ_0:

$$t\mathcal{E}_C^{\phi_0} = \{X \in \mathcal{L} \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \mathcal{D}(\phi_0)\}$$

The property $tGNDC_{\lhd}^{\alpha}$ is defined as follows:

Definition 15 *S is $tGNDC_{\lhd}^{\alpha}$* iff $\forall X \in t\mathcal{E}_C^{\phi_0} : (S||X)\backslash C \lhd \alpha(S)$ *where $\lhd : \mathcal{L} \rightarrow \mathcal{L}$ is a timed behavioural relation between processes and $\alpha : \mathcal{L} \rightarrow \mathcal{L}$ is a function between processes defining the property specification for S as the process $\alpha(S)$.*

As for the case of GNDC, it has been shown that ([GLM03]), for the analysis of safety properties in the timed-trace setting, it is possible to prove the existence of a most general intruder $(tTop_{ttrace}^{C})_{\phi}$, acting as its companion in the non timed setting. Moreover, $(tTop_{ttrace}^{C})_{\phi}$ can let time pass, by performing $tick$ actions. Again, the timed traces of $(tTop_{ttrace}^{C})_{\phi}$ include those of any X belonging to the set $t\mathcal{E}_C^{\phi_0}$, [GLM03].

Thus, the following corollary holds:

Corollary 2 *For every function $\alpha : \mathcal{L} \rightarrow \mathcal{L}$, a process S is $tGNDC_{\leq_{ttrace}}^{\alpha}$* iff $(S||(tTop_{ttrace}^{C}) \setminus C \leq_{ttrace} \alpha(S)$.

We may define several security properties through the t$GNDC$ schema, ***e.g.***, see [GLM03]. For instance, timed secrecy expresses that a certain message m is not known by the intruder within a certain amount of time, say at least n units of time. A specification α_{tSec} dealing with timed secrecy could be the following:

$$pub(m) = \overline{public}(m).\mathbf{0'} + tick.pub(m)$$
$$\alpha_{tSec} = tick_1 \ldots tick_n.(pub(m))$$

[1] Actually, there is another constraint that imposes that the enemy must eventually let time pass. This is however not useful for safety properties we are going to study in this paper and so it has been omitted for the sake of simplicity.

where we assume *public* the unique not restricted channel. α_{tSec} let n units of time pass and then behaves like *pub(m)*, *i.e.*, it could either sends message *m* over channel *public* or it could let one unit of time pass and then it eventually sends *m*.

Note that the GNDC theory is now a well established approach for security analysis and it was developed for non-deterministic, probabilistic, real time and cryptographic frameworks, *e.g.*, see [ABG04, FGM03, FM99, GLM03, GM03].

Chapter 3

The Multicast Chapter

Formal models and analysis of secure multicast and wireless communication

3.1 Introduction

MULTICAST COMMUNICATION AND SECURITY ISSUES. With the wide use of Internet, the popularity of multicast has grown considerably. Examples include live-broadcasts, digitized audio and video, news feeds, stock quotes, multi-party video games, multi-party video conferences, data applets, software updates.

Dealing with multicast communication means, in the terminology currently present in the literature, dealing with digital streams, *i.e.*, long (potentially infinite) sequence of bits. The stream is typically sends from one sender to an established set of receivers.

Given that network security threats have flourished as well, increasing trend to distribute streamed data over the Internet must provide sufficient security guarantees. In particular, so called *stream signature protocols* were born with the intent to efficiently solve the problem to sign digital streams. This class of protocols, designed for open architectures, make usually use hashing techniques and a thrifty use of standard digital signatures to ensure the authentication of the sender and the integrity of the stream.

Indeed, two of the main challenges of securing multicast communication are authentication and integrity. The first challenge deals with the problem of certainly identifying the identity of the sender of the stream. The second challenge deal with enabling receivers of multicast data to verify that the received data was

not modified en-route. These problems become more complex in common settings where other receivers are not trusted and where lost packets are not re-transmitted.

In some cases, secrecy requirements are also due. Let the reader think, for example, to the management of data in a *pay per view* environment: only a restricted group of authorized users must have the ability to consume the stream. Thus, group-encryption techniques are enabled, such as only an established group of users, at a certain time, is able to read the received data.

FORMAL VERIFICATION OF SECURE MULTICAST. Along with the development of schemes for secure multicast, the use of formal techniques for their analysis represents an interesting challenge because of the diversity of such protocols from standard cryptographic schemes. Indeed, two peculiarities are: *i*) a sender broadcasts a continuous (and possibly unbounded) stream of messages to a possibly unbounded set of receivers; *ii*) receivers use information retrieved in earlier packets to legitimate later packets or vice-versa. Some particular scenarios may include mobility, *i.e.*, participants through the protocol can move and change dynamically.

Thus, such a formal analysis have recently raised an interest among researchers. Some proposals have been given in the past few years. In [Arc02], Archer states a formal analysis based on model checking techniques (*i.e.*, checking all the reachable states of a system with respect to the fulfillment of a certain property) is not feasible. In her opinion, this is because "an infinite state system is required to represent the inductive relationship between an arbitrary n-th packet and the initial packet". Instead, she exploits theorem proving techniques to analyze the basic version of a well known stream authentication protocol (the TESLA protocol, [PCST01]). On the other hand, in [BL02] Broadfoot and Lowe show their successful results derived applying model checking techniques on TESLA [PCST01], motivating, even though informally, several steps of the analysis. In particular, they have shown how to build a finite model of TESLA, despite the possibly unbounded stream of messages (and cryptographic keys) broadcasted by the sender.

The analysis approach we are going to show throughout this chapter is quite different from what has been proposed in the literature and focuses its attention on the verifiability of a system with an arbitrary number of components (like the case of stream signature protocols). In particular, it will be shown how to apply a compositional principle allowing us to safely compose processes, in such a way that the overall system preserves the security properties that each subsystem separately enjoys.

WIRELESS COMMUNICATION AND SECURITY ISSUES. Along with securing multicast communication and, consequently, verifying the proposed schemes, researchers have recently addressed the quest for securing (and verifying the security properties of) wireless communication. Indeed, in the following we are also going to show a study of some security aspects of a subclass of wireless networks, *i.e.*, the wireless sensor networks. The attention will be focused on a construction where time plays an essential role.

By means of a wireless network, objects may interact with each other even without the presence of a wired architecture. In particular, wireless systems consisting of mobile nodes self-organizing in temporary routing topologies form wireless *ad hoc* networks. These networks are extremely useful in case of rescue operations in remote areas or disaster recovery operations.

Various issues in the world of the *ad hoc* networks are greatly influenced by timed relationships (*e.g.*, whichever the media access control protocol may be, real-time and temporal synchronization constraints characterize the communications between the hosts). The interested reader is referred to [Rom01] for a fully detailed survey about real-time issues in *ad hoc* networks.

A wireless sensor network is typically composed of hundreds (or even thousands) of sensors, (up to) cubic millimeters devices provided with autonomous sensing, computation and communication. The network is coordinated in a distributed mode in order to collect information on their surroundings. Sensor networks applications are expected to range over many fields, from home applications like automation and smart environments to military uses (monitoring equipment and ammunitions, battlefield management, etc.). Sensors may be used in a building for heating and air conditioning control as well as in a hospital for medical monitoring (*e.g.*, drugs administration or telemonitoring of physiological conditions of the patients), not to mention environmental monitoring, such as the detection of possible fires in a forest.

The development of secure communication between sensors represents a new challenge with respect to standard solutions. Indeed, sensors already have to cope with severe constraints in terms of power consumption, bandwidth and storage and they may not have the resources to perform cryptographic operations in their completeness. Standard solutions developed for conventional computers cannot be applied, hence new schemes have been proposed and surveys have been carried out (see, *e.g.*, [PST+02, LEH03]). A particular complexity is required in [PST+02], where temporal constraints occur, since a time synchronization is needed between a base station and the sensors in the network.

THREAD OF THE CHAPTER. Thus, thread of this chapter is to show the formal capability of capturing new security features in both stream signature protocols and sensors protocols (with real time requirements). The analysis will be given by means of detailed case studies and by exploiting two compositional principles.

COMPOSITIONAL STRATEGY. A compositional principle gives sufficient conditions to conclude that the composition of two (or more) processes satisfies the composition of two (or more) properties, provided that the single processes satisfy the single properties. As an example, such a principle could work as follows: in order to check if a system $P||Q$ satisfies a formula $f_1||f_2$, it is enough to check whether both P satisfies f_1 and Q satisfies f_2. (Notation $||$ represents the parallel composition of subcomponents or subformulas, see also Section 2.3.1, Chapter 2). The existence of such a principle would be particularly appealing for the target of our analysis. Indeed, the state-space of the system $P||Q$ is usually considerably bigger than those of P and Q, separately. Above all, it would help in analyzing systems with a possibly unbounded number of components. Indeed, let the reader consider the parallel composition of equal processes P:

$$\overbrace{P||\ldots||P}^{n}$$

To prove that the overall system enjoys $f* = f||\ldots||f$ (for whatever n) it is sufficient to prove that P enjoys f.

Among the security properties we have briefly presented at the beginning of the chapter, we will mainly deal with the property of integrity, *i.e.*, a sort of robustness against packet modification. For the analysis property, two compositional principles will be exploited.

The first principle was first introduced in [GLM03] for the Generalized Non Deducibility on Compositions scheme of properties, (GNDC for short), defined in [FGM00a, FM99]. In turn, the scheme (reminded in Section 2.4 of Chapter 2) is based on the seminal notion of non-interference, [GM82]. The first principle was successfully applied in [GMPV03a] to verify an instance of a pioneer protocol for signing digital streams, see [GR01] and Subsection 3.2.1 of this chapter. It was also successfully applied in [MPV03] to verify an instance of a stream signature protocol dealing with packet loss, see [PCTS00] and Subsection 3.2.2 of this chapter.

The second principle was first introduced in [GMPV03b] within a formal framework aimed at verifying timed security properties, *i.e.*, security properties

whose fulfilment is based on timed conditions. In its turn, the timed formal framework, namely the timed-Generalized Non Deducibility on Compositions (tGNDC for short) has been introduced in [GLM03]. Part of Section 2.4 in Chapter 2 has been devoted to remind the reader of the tGNDC schema.

CASE STUDIES. The analyzed case studies are three significant protocols aimed at guaranteeing data integrity to the information flow from a sender to a set of receivers, where integrity means, informally, that the information accepted by each receiver is exactly what the sender has intended.

Namely, the protocols are: 1) the Gennaro-Rohatgi protocol [GR01], a pioneer protocol introduced in 1997 to sign digital streams; 2) the Efficient Multi-chained Stream Signature protocol (EMSS) proposed in [PCTS00]. This stream signature protocol implements a significant improvement with respect to the Gennaro-Rohatgi protocol, since it guarantees some robustness against packet loss, whereas the Gennaro-Rohatgi protocol does not; 3) finally, the μTESLA ("micro" Timed Efficient Stream Loss-tolerant Authentication), a protocol to provide authenticated broadcast in wireless sensor networks environments.

A detailed formal model of all of them will be given in a subsequent section, by exploiting the formal process algebras whose syntax and semantics have been concisely presented in Section 2.3.1 of Chapter 2.

CONTRIBUTIONS. The main contributions of this chapter are the following.

i) We develop a compositional analysis technique able to deal with multicast protocols has been developed ([GMPV03a, MPV03]).

ii) In those two investigations, we are not able to manage protocols with time-dependent security properties. Thus, we give a new compositional principle for dealing with timed security properties (as timed secrecy and timed integrity). The principle is new with respect to the one given in [GLM03, GMPV03a, MPV03] and it has been first introduced in [GMPV03b].

iii) By means of a real-time process algebra, ($tCryptoSPA$ in the Preliminaries) we outline a formal framework ([GMPV03b]) for modeling wireless communication.

iv) We formally model and analyze three relevant proposals for authenticating data streams. To the best of our knowledge, this is the first attempt to prove some of the security properties of those protocols (by means of compositional rules).

v) Starting from modeling the basic scheme of Gennaro and Rohatgi, passing through protocols dealing with packet loss, concluding with a time-dependent security wireless protocol, the proposed analysis aims at allowing the modeling and formal validation of a set of multicast and wireless protocols.

vi) Contrary to previous work in the area, *e.g.*, [Arc02, BL02], the proposed analysis is able to check a specification with an unbounded number of components. (The target of our analysis however being different from TESLA, the protocol analyzed in [Arc02, BL02]).

SUMMARY. The chapter is organized as follows. Section 3.2 deals with the informal description and the formal model of the three mentioned case studies. Section 3.3 introduces the concept of stability of a process and it illustrates the first compositional principle, to establish if a system enjoys a security property defined by means of GNDC. Section 3.4 introduces the concept of time dependent stability of a process and it illustrates the second compositional principle, to establish if a system enjoys a timed security property defined by means of tGNDC. Section 3.5 shows hot to apply the results shown in Section 3.3 to successfully prove the correctness of the EMSS construction in terms of packets' integrity. Section 3.6 shows hot to apply the results shown in Section 3.4 to successfully prove the correctness of the μTESLA construction in terms of packets' timed integrity.

3.2 Modeling multicast communication

In this section, we present and formally model three security protocols for multicast communication. Basically, all of them aim at ensuring integrity and authenticity of the so called *digital streams*, *i.e.*, long (potentially infinite) sequence of bits. Typically, communication involve one sender and an arbitrary number of receivers.

A formal analysis of the integrity property of two of the three protocols will be carried out in the following sections.

3.2.1 The Gennaro-Rohatgi protocol

In [GR01], Gennaro and Rohatgi developed a mechanism to sign digital streams. They aim at assuring a receiver that the information he received is exactly what the sender has intended.

Applications that deal with streams are typically digitized audio and video, data feeds, applets. This kind of applications requires the user to consume the data it receives at almost the input rate, without excessive delay. For this reason, signing digital streams represents a different problem compared with the signature of finite messages. Traditional digital signature schemes do not fit properly

because they require the receiver to process the entire message in order to verify the signature.

It is our opinion that the Gennaro-Rohatgi protocol should be considered paradigmatic, being essentially, in its 1997 version, one of the first proposals to efficiently solve the problem to sign digital streams. Efficient cryptographic solutions (*i.e.*, fast to be computed and verified, with respect to the time in which these authors made the proposal) have been adopted in the protocol to allow the entities at stake to minimize their communication and computation overhead.

The authors present two solutions to the problem, distinguishing two cases: i) the off-line case: a finite stream which is entirely known to the sender (*e.g.*, a movie); ii) the on-line case: a potentially infinite stream not known in advance to the sender (*e.g.*, a live broadcast for news feed).

We model the off-line scheme below. For details about the on-line scheme, the reader is referred to [GR01, GMPV03a] and Appendix A.

The off-line scheme relies on the basic idea to divide the stream into blocks and to add cryptographic information in each block such that receivers use information retrieved in earlier blocks to legitimate later blocks.

We first use an intuitive notation usually found in literature. We consider a set of agents able to receive messages. With the following notation,

$$label \quad c_j \quad A \rightarrow B \quad : \quad msg$$

we represent the transmission of message msg from a sender A to a receiver B. c_j is the j-th communication channel, $label$ is the name of msg.

Thus, let $\{b_i\} \in Msgs$ be the set of meaningful payloads, $i = 1 \ldots l$[1]. Then, the protocol for the off-line case is:

$$\begin{array}{lllllll}
\text{Block } b_0^{'} & c_0 & S & \rightarrow & R & : \{h(b_1^{'})\}_{sk(S)} \\
\text{Block } b_i^{'} & c_i & S & \rightarrow & R & : b_i, h(b_{i+1}^{'})\ i = 1..l-1 \\
\text{Block } b_l^{'} & c_l & S & \rightarrow & R & : b_l
\end{array}$$

It exploits the technique of embedding the hash of the following block in the current block. Bootstrapping integrity of the digital stream is obtained by applying a single traditional signature in combination with hash chaining.

$h(m)$ is the digest of m after applying the hash function; $\{m\}_{sk(S)}$ is message m digitally signed by the sender's private key $sk(S)$.

The sender S first divides the stream to be sent into l blocks. Then, S generates the digital signature on the hash of the first block (Block $b_0^{'}$). After verification

[1]It is assumed that the sender's private key $sk(S)$ does not occur in the set $\{b_i\}$

of the signature the receiver knows what the hash of the first block should be and then it starts receiving the full stream (blocks $b_i^{'}$). When the receiver receives the first block $b_1^{'}$, it computes its hash and checks the hash against what the signature was verified upon. The other blocks consist of an authentication chain, in which each block contains the hash of the subsequent block. Note that embedding the hash of the subsequent block implies that the sender knows the stream in advance, hence the non feasibility of this construction for applications like live broadcasts.

It is worth noticing that in the original paper [GR01], the first block contains an encoding of the length of the stream. The structure of the first block is here simplified (without however affecting the results of the analysis that will follow). Furthermore, we assume the receiver knows in advance the number of blocks in which the stream is divided. It is also worth noticing that to avoid replay attacks when executing multiple runs of the protocol one can simply include nonces in the digitally signed block.

Crypto-CCS specification of the Gennaro-Rohatgi protocol

To formally specify the protocol, the sender and the receiver are modelled as Crypto-CCS processes. A suitable inference system that is used to model this digital stream signature protocol is shown in Fig. 3.1. Rule $(pair)$ builds the pair of two messages x and y; rules (fst) and (snd) return the components of a pair; rule $(sign)$ allows message x to be digitally signed by applying the secret key $sk(y)$ of agent y; rule (ver) allows a digital signature $\{x\}_{sk(y)}$ to be verified by applying the public key of signer y, $pk(y)$; rule $(hash)$ allows an agent to apply a one-way hash function to message x and obtain digest $h(x)$.

In the following, each conclusion of an inference construct is a message variable. With notation x_m we mean "variable x should contain message m".

$$\frac{x \quad y}{Pair(x,y)}(pair) \qquad \frac{Pair(x,y)}{x}(fst) \qquad \frac{Pair(x,y)}{y}(snd)$$

$$\frac{x \quad sk(y)}{\{x\}_{sk(y)}}(sign) \qquad \frac{\{x\}_{sk(y)} \quad pk(y)}{x}(ver) \qquad \frac{x}{h(x)}(hash)$$

Figure 3.1: Inference system for the Gennaro-Rohatgi protocol.

$Sender_0 \doteq$
$[b_1^{'} \vdash_{hash} x_{h(b_1^{'})}]$ *Compute hash of next block*
$[x_{h(b_1^{'})} \quad sk(S) \vdash_{sign} x_{b_0^{'}}]$ *Sign computed hash*
$c_0!x_{b_0^{'}}.Sender_1$ *Output* $b_0^{'}$ *and go to next state*

$Sender_i \doteq$
$[b_{i+1}^{'} \vdash_{hash} x_{h(b_{i+1}^{'})}]$ *Compute hash of next block*
$[b_i \quad x_{h(b_{i+1}^{'})} \vdash_{pair} x_{b_i^{'}}]$ *Pair payload and hash next block*
$c_i!x_{b_i^{'}}.Sender_{i+1}$ *Output* $b_i^{'}$ *and go to next state*

$Sender_l \doteq$
$c_l!b_l.\mathbf{0}$ *Output last block and stop*

The sender process builds the initialization block $b_0^{'}$ (more precisely, he builds a variable containing $b_0^{'}$) to bootstrap the chain: by means of inference rules $hash$ and $sign$ in Fig. 3.1 the sender computes block $b_0^{'}$, sends it on communication channel c_0 and travels to the next state $Sender_i$. It now sends payloads b_i together with hashed blocks $h(b_{i+1}^{'})$ until the last state l is reached.

The receiver process is parameterized by the hashed blocks he receives from the sender (more precisely, variables that should contain the hashed blocks).

$Receiver_0(null) \doteq$	
$c_0?x_{b'_0}.$	*Receive initial block*
$[x_{b'_0} \quad pk(S) \vdash_{ver} x_{h(b'_1)}]$	*Verify signature*
$Receiver_1(x_{h(b'_1)})$	*Go to next state*
$Receiver_i(x_{h(b'_i)}) \doteq$	
$c_i?x_{b'_i}.$	*Receive* $i-th$ *block*
$[x_{b'_i} \vdash_{hash} x_{h(b'_{MYi})}]$	*Compute my hash* $h(b'_{MYi})$
$[x_{h(b'_i)} = x_{h(b'_{MYi})}]$	*Compare hash*
$[x_{b'_i} \vdash_{fst} x_{b_i}]$	*Extract payload*
$c_{out_i}!x_{b_i}.$	*Send payload to application level*
$[x_{b'_i} \vdash_{snd} x_{h(b'_{i+1})}]$	*Extract hash of next block and*
$Receiver_{i+1}(x_{h(b'_{i+1})})$	*go to next state*
$Receiver_l(x_{h(b'_l)}) \doteq$	
$c_l?x_{b'_l}.$	*Receive last block*
$[x_{b'_l} \vdash_{hash} x_{h(b'_{MYl})}]$	*Compute my hash* $h(b'_{MYl})$
$[x_{h(b'_l)} = x_{h(b'_{MYl})}]$	*Compare hash*
$c_{out_l}!x_{b'_l}.\mathbf{0}$	*Send block to application level and stop*

In the initial state the receiver aims at verifying the digital signature (we assume he has previously retrieved the public key $pk(S)$ corresponding to the private key of the supposed sender). Then, it travels to the next state $Receiver_i(x_{h(b'_1)})$, by maintaining history of the (supposed) next hashed block $h(b'_1)$. Acceptance of the subsequent blocks is conditioned to the successful outcome of the equality tests between the hash it maintains as a parameter and the hash it computes from what it has presently received, respectively $x_{h(b'_i)}$ and $x_{h(b'_{MYi})}$. The successful outcome of the equality test is here modeled by imaging that the receiver sends the meaningful payload contained in x_{b_i} to the application level to consume it. This sending operation is over channel c_{out_i}. The receiver then extracts the supposed hash of the block to be received immediately later. This mechanism is repeated until the reception of the l-th block. Whether the verification of the signature in the initial state or the equality tests in subsequent states do not succeed the receiver should abort.

Extending the model to multiple receivers.

Extending the model to the treatment of multicast and broadcast communication (*i.e.*, , by allowing a potentially unbounded number of receivers) is as follows: a new process MB is added, that is responsible for potentially sending each block an unbounded number of times in order to simulate a one-to-many (one-to-all) sending typical of a multicast (broadcast) communication. The new process is parameterized by the block the sender is to multicast (or broadcast).

$$MB_i(x_{b_i'}) \doteq c_i!x_{b_i'}.MB_i(x_{b_i'})$$

Thus, in the light of this new process, the specification for the sender process can be re-written as follows:

$Sender_0 \doteq$
$[b_1' \vdash_{hash} x_{h(b_1')}]$
$[x_{h(b_1')} \quad sk(S) \vdash_{sign} x_{b_0'}]$
$(Sender_1 || MB_0(x_{b_0'}))$ *Output* b_0' *and go to next state*

$Sender_i \doteq$
$[b_{i+1}' \vdash_{hash} x_{h(b_{i+1}')}]$
$[b_i \quad x_{h(b_{i+1}')} \vdash_{pair} x_{b_i'}]$
$(Sender_{i+1} || MB_i(x_{b_i'}))$ *Output* b_i' *and go to next state*

$Sender_l \doteq MB_l(b_l)$ *Output* b_l *and go to next state*

3.2.2 The EMSS protocol

Digital streams are usually sent over UDP, the User Datagram Protocol, [Pos80]. UDP is considered to be an unreliable transport protocol, *i.e.*, when UDP sends packets over a network, it just sends them and forgets about them. This does not mean that UDP is ineffective, only that it does not handle reliability of the communication. If a stream is received incomplete, we would still like to be able to prove the integrity of all the packets that were not lost.

Along with the pioneer protocol modelled in the previous section, protocols dealing with the problem of securing streamed data over channels with packet loss have been recently proposed, [PCTS00, PCS02, GM01]. They all can be basically considered as valuable extensions of the Gennaro-Rohatgi constructions.

In particular, in [PCTS00], Perrig *et al.* presented the Efficient Multi-chained Stream Signature (EMSS) protocol to sign digital streams. EMSS exploits a combination of hash functions and digital signatures and–contrary to previous proposals [GR01]–achieves (some) robustness against packet loss.

The basic idea of EMSS is the following: a hash of packet P_{i-1} is appended to packet P_i, whose hash is in turn appended to packet P_{i+1} and so on. A signature packet, containing the hash of the final data packet along with a signature, is sent at the end of the stream. To achieve robustness against packet loss (the event of one or more packets loss would break the chain) each packet contains multiple hashes of previous packets and the signature packet signs hashes of multiple packets. [PCTS00] uses both deterministic and random distribution of hashes per packet.

Here we focus on a specific instance of the EMSS, viz. the deterministic (1,2) schema, where packet P_i contains hashes of packets $i-1, i-2$ and whose hash is contained in packets $i+1, i+2$. After an initial phase, each packet P_i contains a meaningful payload m_i[2] together with the hashes $h(P_{i-1})$ and $h(P_{i-2})$ of the previous two packets sent. Packets are sent over channels $c_i, 0 \leq i \leq last$ from a sender S to a set of receivers $\{R_n \mid n \geq 1\}$. The end of a stream is indicated by a signature packet P_{sign} over channel c_{sign}, containing the hashes of the final two packets, along with a digital signature. The protocol can formally be described as follows.

$$
\begin{array}{lllllll}
\text{Packet } P_0 & c_0 & S & \rightarrow & \{R_n\} & : m_0, null, null & \\
\text{Packet } P_1 & c_1 & S & \rightarrow & \{R_n\} & : m_1, h(P_0), null & \\
\text{Packet } P_i & c_i & S & \rightarrow & \{R_n\} & : m_i, h(P_{i-1}), h(P_{i-2}) & 2 \leq i \leq last
\end{array}
$$

Let P_{last} be the last packet of the stream. Upon sending P_{last} a signature packet is sent:

$$
\text{Sign-Pack} \quad P_{sign} \quad c_{sign} \quad S \quad \rightarrow \quad \{R_n\} \quad : \{h(P_{last}), h(P_{last-1})\}_{sk(S)}
$$

A packet P_i is said to be verifiable if there exists a path (in terms of hash chains) from P_i to the signature packet. Given a set of verifiable packets, we intend to prove the correctness of the construction in terms of packet integrity, *i.e.*, to assure a receiver that the information it received is exactly what the sender has originally intended. For the analysis, see Section 3.5. Furthermore, see the following chapter, Chapter 4, for a translation of EMSS in terms of Team Automata. By taking into account the same case study, one can compare their expressiveness in describing multicast communication and reliability of the transmission.

[2] We assume the sender's private key $sk(S)$ cannot be deduced from the set of messages $\{m_i\}$.

Crypto-CCS specifications of the (1,2) EMSS.

We present the Crypto-CCS specifications of the (1,2) scheme of the EMSS protocol.

We remind that the whole formalization, in particular the way a receiver process acts, is based on implementative choices of the authors since some details are not explicitly given in [PCTS00].

A suitable inference system that is used to model EMSS is shown in Fig. 3.2. Rule $(tuple)$ builds the tuple of three messages x, y and z; rules $(1-st)$, $(2-nd)$ and $(3-rd)$ return, respectively, the first, second and third component of a tuple; rules $(sign)$, (ver) and $(hash)$ are the same as in the inference system for the Gennaro-Rohatgi protocol.

The sender process is parameterized by variables containing the hashes it should insert in the following packet. As in the formalization of the Gennaro-Rohatgi protocol, with notation x_m we mean "variable x should contain message m".

$S_0(0,0) \doteq$
$[m_0 \vdash_{tuple} x_{P_0}]$ — *Create tuple*
$[x_{P_0} \vdash_{hash} x_{h(P_0)}]$ — *Compute hash of* P_0
$(S_1(x_{h(P_0)},0)||MB_0(x_{P_0}))$ — *Output* P_0 *and go to next state*

$S_1(x_{h(P_0)},0) \doteq$
$[m_1 \quad x_{h(P_0)} \vdash_{tuple} x_{P_1}]$ — *Create tuple*
$[x_{P_1} \vdash_{hash} x_{h(P_1)}]$ — *Compute hash of* P_1
$(S_2(x_{h(P_1)},x_{h(P_0)})||MB_1(x_{P_1}))$ — *Output* P_1 *and go to next state*

$S_i(x_{h(P_{i-1})},x_{h(P_{i-2})}) \doteq$
$[m_i \quad x_{h(P_{i-1})} \quad x_{h(P_{i-2})} \vdash_{tuple} x_{P_i}]$ — *Create tuple*
$[x_{P_i} \vdash_{hash} x_{h(P_i)}]$ — *Compute hash of current packet*
$(S_{i+1}(x_{h(P_i)},x_{h(P_{i-1})})||MB_i(x_{P_i}))$ — *Output* P_i *and go to next state*

$S_{sign}(x_{h(P_{last})},x_{h(P_{last-1})}) \doteq$
$[x_{h(P_{last})} \quad x_{h(P_{last-1})} \vdash_{tuple} x_t]$ — *Create tuple of final hashes*
$[x_t \quad sk(S) \vdash_{sign} x_{P_{sign}}]$ — *Sign the tuple*
$MB_{sign}(x_{P_{sign}})$ — *Output the signature packet*

Again, the special process MB is responsible for potentially sending each packet an unbounded number of times, in order to simulate a one-to-many (one-to-

$$\frac{z \;\; y \;\; z}{(x,y,z)}(\vdash_{tuple}) \quad \frac{(x,y,z)}{x}(\vdash_{1-st}) \quad \frac{(x,y,z)}{y}(\vdash_{2-nd}) \quad \frac{(x,y,z)}{z}(\vdash_{3-rd})$$

$$\frac{x \quad sk(y)}{\{x\}_{sk(y)}}(\vdash_{sign}) \qquad \frac{\{x\}_{sk(y)} \;\; pk(y)}{x}(\vdash_{ver}) \qquad \frac{x}{h(x)}(\vdash_{hash})$$

Figure 3.2: Inference system for EMSS.

all) sending. The process is parameterized by the packet the sender is to multicast (or broadcast).

$$\begin{aligned} MB_i(x_{P_i}) &\doteq c_i!x_{P_i}.MB_i(x_{P_i}) \quad 0 \leq i \leq last \\ MB_{sign}(x_{P_{sign}}) &\doteq c_{sign}!x_{P_{sign}}.MB_{sign}(x_{P_i}) \end{aligned}$$

Among the set of receivers, each process behaves in the same way. The generic receiver process at step i is parameterized by: 1) the two last packets it received (let them be P_{j_1}, P_{j_2}) - over an ideal channel, without packet loss, we have that $P_{j_1} = P_{i-1}$ and $P_{j_2} = P_{i-2}$; 2) a tuple $tup^{i-1}_{\{m_j\}}$. $tup_{\{m_j\}}$ consists of the ordered sequence of payloads among $\{m_j\}_{j=0,1,...last}$ whose corresponding packets' hashes $h(P_j)$ the receiver was able to check[3]. $tup^{i-1}_{\{m_j\}}$ is the tuple updated at step i, by inserting either $x_{m_{i-2}}$ or $x_{m_{i-3}}$. $tup^{i-1}_{\{m_j\}}$ could be either $(x_{m_{i-2}}, tup^{i-1}_{\{m_j\}})$ or $(x_{m_{i-3}}, tup^{i-1}_{\{m_j\}})$ (also, it may remain unchanged). Similarly, $tup^{last}_{\{m_j\}}$ may either be $(x_{m_{last}}, tup^{last}_{\{m_j\}})$ or $(x_{m_{last-1}}, tup^{last}_{\{m_j\}})$ or, unchanged, $tup^{last}_{\{m_j\}}$.

The unreliability of the transmission over UDP is modeled by considering that process Rec non deterministically chooses whether to receive a packet or not. Finally, we assume that the signature packet P_{sign} is always received (this is likely since in the original protocol multiple copies of the signature packets are sent).

[3]For the sake of readability we assume the receiver may infer the sequence number of a packet by simply observing the packet itself. Otherwise, we should arrange the receiver with more parameters or arrange a "sequence number" field in the packet structure and let the receiver retrieve it. This could introduce a too clumsy notation.

$Rec_0(0,0,0) \doteq$
$Rec_1(0,0,0) +$ — *Packet loss : go to next state, otherwise*
$(c_0?x_{P_0}.$ — *Receive initial packet*
$Rec_1(x_{P_0},0,0))$ — *Go to next state*

$Rec_1(0,0,0) \doteq$
$Rec_2(0,0,0) +$ — *Packet loss : go to next state, otherwise*
$(c_1?x_{P_1}.$ — *Receive packet P_1*
$Rec_2(x_{P_1},0,0))$ — *Go to next state*

$Rec_1(x_{P_0},0,0) \doteq$
$Rec_2(0,x_{P_0},0)+$ — *Packet loss : go to next state, otherwise*
$(c_1?x_{P_1}.$ — *Receive packet P_1*
$[x_{P_1} \vdash_{2-nd} x_{h(P_0)}]$ — *Extract hash of previous packet P_0*
$[x_{P_0} \vdash_{hash} x_{h_{MY}(P_0)}]$ — *Compute my hash $h_{MY}(P_0)$*
$[x_{h(P_0)} = x_{h_{MY}(P_0)}]$ — *Compare the hashes*
$([x_{P_0} \vdash_{1-st} x_{m_0}]$ — *IF equal : extract previous payload*
$Rec_2(x_{P_1},x_{P_0},x_{m_0})$ — *Update parameters and go to next state*
$);\mathbf{0}$ — *ELSE abort*
$)$

$Rec_i(x_{P_{j_1}},x_{P_{j_2}},tup^{i-1}_{\{m_j\}}) \doteq$
$Rec_{i+1}(x_{P_{j_1}},x_{P_{j_2}},tup^{i-1}_{\{m_j\}}) +$ — *Packet loss : go to next state, otherwise*
$(c_i?x_{P_i}.$ — *Receive packet P_i*
$([j_1 = i-1]$ — *Was P_{i-1} received?*
$Rec'_i(x_{P_i},x_{P_{i-1}},tup^{i-1}_{\{m_j\}});$ — *Go to Rec'_i; otherwise*
$([j_2 = i-2]$ — *Was P_{i-2} received?*
$Rec''_i(x_{P_i},x_{P_{i-2}},tup^{i-1}_{\{m_j\}})$ — *Go to Rec''_i; otherwise*
$)$
$); Rec_{i+1}(x_{P_i},x_{P_{j_1}},tup^{i-1}_{\{m_j\}})$ — *Go to next state :*
$)$ — *P_{1-1} and P_{i-2} were not received*
$Rec'_i(x_{P_i},x_{P_{i-1}},tup^{i-1}_{\{m_j\}}) \doteq$
$[x_{P_i} \vdash_{2-nd} x_{h(P_{i-1})}]$ — *Extract $h(P_{i-1})$ from P_i*
$[x_{P_{i-1}} \vdash_{hash} x_{h_{MY}(P_{i-1})}]$ — *Compute my hash $h_{MY}(P_{i-1})$*
$[x_{h_{MY}(P_{i-1})} = x_{h(P_{i-1})}]$ — *Compare the hashes*
$([x_{P_{i-1}} \vdash_{1-st} x_{m_{i-1}}]$ — *IF equal : extract m_{i-1} from P_{i-1}*
$Rec_{i+1}(x_{P_i},x_{P_{j_1}},(x_{m_{i-1}},tup^{i-1}_{\{m_j\}}))$ — *Update parameters and go to next state*
$);\mathbf{0}$ — *ELSE : abort*

$Rec''_i(x_{P_i}, x_{P_{i-2}}, tup^{i-1}_{\{m_j\}}) \doteq$

$[x_{P_i} \vdash_{3-rd} x_{h(P_{i-2})}]$	*Extract* $h(P_{i-2})$ *from* P_i
$[x_{P_{i-2}} \vdash_{hash} x_{h_{MY}(P_{i-2})}]$	*Compute my hash* $h_{MY}(P_{i-2})$
$[x_{h_{MY}(P_{i-2})} = x_{h(P_{i-2})}]$	*Compare the hashes*
$([x_{P_{i-2}} \vdash_{1-st} x_{m_{i-2}}]$	*IF equal : extract* m_{i-2} *from* P_{i-2}
$Rec_{i+1}(x_{P_i}, x_{P_{j_1}}, (x_{m_{i-2}}, tup^{i-1}_{\{m_j\}}))$	*Update parameters and go to next state*
$); \mathbf{0}$	*ELSE : abort*

$Rec_{sign}(x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}}) \doteq$

$c_{sign}?x_{P_{sign}}.$	*Receive signature packet*
$Rec^*_{sign}(x_{P_{sign}}, x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}})$	*Go to intermediary state* Rec^*_{sign}

$Rec^*_{sign}(x_{P_{sign}}, x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}}) \doteq$

$[x_{P_{sign}} \quad pk(S) \vdash_{ver} x_{ver}]$	*Verify the signature*
$[j_1 = last]$	*Was* P_{last} *received?*
$Rec'_{sign}(x_{ver}, x_{P_{last}}, tup^{last}_{\{m_j\}});$	*If so, go to* Rec'_{sign}*; otherwise*
$([j_2 = last - 1]$	*Was* P_{last-1} *received?*
$Rec''_{sign}(x_{ver}, x_{P_{last-1}}, tup^{last}_{\{m_j\}});$	*If so, go to* Rec''_{sign}*; otherwise*
$(c_{app}!tup^{last}_{\{m_j\}}.\mathbf{0})$	P_{last} *and* P_{last-1} *were not received.*
$)$	*Send the stream of verifiable payloads to the application level*

$Rec''_{sign}(x_{ver}, x_{P_{last-1}}, tup^{last}_{\{m_j\}}) \doteq$

$[x_{ver} \vdash_{2-nd} x_{h(P_{last-1})}]$	*Extract* $h(P_{last-1})$ *from* P_{sign}
$[x_{P_{last-1}} \vdash_{hash} x_{h_{MY}(P_{last-1})}]$	*Compute my hash* $h_{MY}(P_{last-1})$
$[x_{h_{MY}(P_{last-1})} = x_{h(P_{last-1})}]$	*Compare the hashes*
$[x_{P_{last-1}} \vdash_{1-st} x_{m_{last-1}}]$	*IF equal : extract* m_{last-1} *from* P_{last-1}
$c_{app}!(x_{m_{last-1}}, tup^{last}_{\{m_j\}}).\mathbf{0};$	*Send the stream of verifiable payloads*
$\mathbf{0}$	*to the application level and stop; ELSE abort*

$Rec'_{sign}(x_{ver}, x_{P_{last}}, tup^{last}_{\{m_j\}}) \doteq$

$[x_{ver} \vdash_{1-st} x_{h(P_{last})}]$	*Extract* $h(P_{last})$ *from* P_{sign}
$[x_{P_{last}} \vdash_{hash} x_{h_{MY}(P_{last})}]$	*Compute my hash* $h_{MY}(P_{last})$
$[x_{h_{MY}(P_{last})} = x_{h(P_{last})}]$	*Compare the hashes*
$[x_{P_{last}} \vdash_{1-st} x_{m_{last}}]$	*IF equal : extract* m_{last} *from* P_{last}
$c_{app}!(x_{m_{last}}, tup^{last}_{\{m_j\}}).\mathbf{0};$	*Send the stream of verifiable payloads*
$\mathbf{0}$	*to the application level and stop; ELSE abort*

In the final state Rec_{sign} (along with intermediary states $Rec^*_{sign}, Rec'_{sign}, Rec''_{sign}$)

the receiver aims at verifying the digital signature (we assume it has previously retrieved the public key $pk(S)$ corresponding to the private key of the supposed sender). The correct verification of the signature implies the receiver to have guarantees on the integrity of the verifiable payloads. It can now send the stream to the application level to consume it. In our formalization, this is modeled by a scenario where the receiver sends the content of its parameter tuple (the accepted stream) over channel c_{app}. If the verification of the signature in the final state or the equality tests in the previous states do not succeed the receiver should abort.

3.2.3 The μTESLA protocol

In [PST+02], Perrig *et al.* presented μTESLA ("micro" Timed Efficient Stream Loss-tolerant Authentication), a protocol to provide authenticated broadcast in wireless sensor networks environments. [PST+02] considers a scenario where sensors communicate with a base-station connected to the external world. The base station may broadcast to all nodes messages for routing updates, reprogramming, reset requests. The protocol is an extension of the TESLA stream authentication protocol developed in [PCST01] and it was intentionally developed for providing authenticated broadcast for the limited computing environments that are encountered in sensor networks.

In the original TESLA schema, a single sender broadcasts a continuous stream of packets. Receivers may use information in later packets to authenticate earlier packets. Each packet contains a message authentication code (MAC), *i.e.*, a value computed by applying a public algorithm and a secret encryption key to the packet itself. Given a message *m* and an encryption key *k*, we call *mac(m, k)* the message authentication code of *m*. The algorithm is known by all the receivers, while the encryption keys are disclosed by the sender after a certain amount of time. When a receiver receives a key K_i it can use it to compute the MAC from the related packet P_i and compare the computed MAC with that previously received. If the two MACs match, the receiver can consider the packet P_i authentic. To avoid the event that an intruder could use a disclosed key K_i to fake the packet P_i a time synchronization protocol between the sender and the receivers is needed. Then, each receiver will not accept the packet P_i if the sender might have already disclosed the key K_i.

Bootstrapping authentication of the whole scheme is achieved in TESLA by signing the first packet with a regular digital signature scheme. Nevertheless, computation, communication and storage overhead make the use of asymmetric cryptography unfeasible for the net of sensors under investigation. Thus, μTESLA

has been proposed as an optimized extension for sensor networks. It just makes use of MACs. The base-station randomly generates the last MAC key to be used, K_{last}, and derives a key chain by repeatedly applying a publicly known one-way function F to that key, such that $K_i = F(K_{i+1})$. Given the non-reversibility property (at least with high probability) of function *F*, the disclosure of key K_i should not lead to any knowledge of K_{i+1} and subsequent keys.

Receivers' requirements for correctly joining and executing the protocol are: i) they are time synchronized with the base station; ii) they know the disclosure schedule of the MAC keys; iii) they know at least one authenticated key of the key chain, serving as a commitment to the entire chain. A protocol providing time synchronization and one authenticated key has been proposed in [PST+02]. Basically, the base-station shares with each sensor a symmetric secret key K_{SM} and establishes a secure channel over which the exchange of a commitment to the key chain, K_0, and a set of temporal parameters, set_t, takes place[4]. More formally, the initial step of μTESLA is the following:

$$\text{Packet } P_0 \qquad c_0 \quad S \quad \rightarrow \quad \{R_n\} \quad : K_0, set_t, mac(K_0, set_t, K_{SM})$$

where $c_0 \in \{c_i\}_{i \in \mathbb{N}}$, *i.e.*, the set of communication channels, S is the identifier of the sender (*i.e.*, the base station) and $\{R_n\}$ is the set of receivers (*i.e.*, the sensors)[5].

μTESLA is parameterized by the schedule time at which MAC keys are disclosed. For the description of further steps in the protocol we consider a basic formalization, Fig. 3.3, where we suppose that the sender discloses a MAC key with a delay $\delta = 1$, assumed to fall in the interval after that key has been used to compute the MAC. Further, we suppose the sender sends one packet per time interval. Basically, in each time slot a packet and a key packet will be sent, see Fig. 3.3. First of all, each receiver should check the integrity of the received key, say K_i, by verifying it w.r.t. an authenticated commitment (*e.g.*, by checking $K_0 = F^i(K_i)$), then the verified key will be used to verify the integrity of the packet received in the previous time slot.

$$\text{Packet } P_i \qquad c_i \quad S \quad \rightarrow \quad \{R_n\} \quad : m_i, mac(m_i, K_i)\ i \geq 1$$

[4]There are as many symmetric keys as the number of sensors and the communication over channel c_0 is supposed to be a point to point communication. Nevertheless, to simplify our formalization, we assume a unique key and a unique communication. This means to implicitly assume that possible adversaries are not in the set of receivers.

[5]To assure freshness when executing multiple runs of the same sender, one can simply insert nonces in the message authentication code of packet P_0.

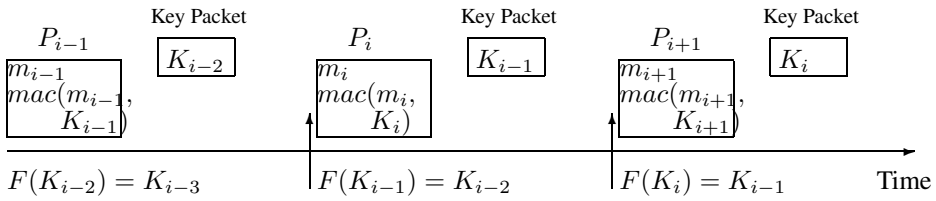

Figure 3.3: A μTESLA instantiation.

Packet P_i consists of a meaningful payload m_i plus the message authentication code computed on m_i with key K_i. We assume that K_{SM} cannot be deduced from the sets $\{m_i\}, \{K_i\}$.

Upon receiving the packet, the sensor stores the packet until its MAC can be verified, *i.e.*, until the sender broadcasts packet disclosing K_i:

$$\text{Key-Packet } KP_i \qquad c_{i+1} \quad S \quad \rightarrow \quad \{R_n\} \quad : K_i$$

The integrity of key K_i can be checked by verifying $K_0 = F^i(K_i)$ (or, equivalently, $K_{i-1} = F(K_i)$). Packets may be lost in transit from the base station to the sensors. In particular μTESLA is tolerant to packet loss in the sense that receivers may still be able to authenticate all the received packets P_i even when the corresponding keys' disclosure packets are lost. Suppose K_j is lost, then a receiver is not able to verify MAC packet P_j. The following key the receiver recovers, let it be K_{j+1}, can be verified w.r.t. a previous authenticated key (*e.g.*, $K_0 = F^{j+1}(K_{j+1})$) and is used to derive K_j, i.e. $K_j = F(K_{j+1})$.

The $tCryptoSPA$ specifications of the μTESLA protocol

We present the $tCryptoSPA$ specification of the basic μTESLA presented in Fig. 3.3.

The fundamental requirement of a time synchronization between a base station and each sensor in μTESLA is naturally captured in $tCryptoSPA$ (Subsection 2.3.1) by its time modeling action $tick$, upon which sender and receivers' processes may synchronize (this allows us to avoid the explicit presence of set_t in packet P_0).

A suitable inference system that is used to model μTESLA is shown in Fig. 3.4. Rule $(one - way)$ allows to apply a one-way hash function F to message m and obtain digest $F(m)$; rule mac computes the message authentication code (MAC)

of a message with a key; rules $(pair)$, (fst) and (snd) are the same as in the inference system for the Gennaro-Rohatgi protocol.

We consider a sender machine with ample resources. It can be parallelized or split into n senders, each of them possibly sending different streams, $\{m_i^j\}_{i\geq 1, 1\leq j\leq n}$. We first present the generic sender process S^j, parameterized by a sequence of MAC keys (tied together by means of a key chain)[6]. We assume the symmetric key K_{SM}, the keys belonging to the key chain and the streams of packets to be different for each process $S^j, 1 \leq j \leq n$[7].

$$
\begin{array}{ll}
S_0^j(K_{SM}^j, K_0^j, K_1^j, \ldots) \doteq & \\
\quad [K_0^j \quad K_{SM}^j \vdash_{mac} y] & \text{Compute MAC} \\
\quad [K_0^j \quad y \vdash_{pair} P_0] & \text{Create packet } P_0 \\
\quad B_0^j(P_0) & \text{Start to broadcast } P_0
\end{array}
$$

$$
\begin{array}{ll}
S_1^j(K_0^j, K_1^j, \ldots) \doteq & \\
\quad [m_1^j \quad K_1^j \vdash_{mac} x] & \text{Compute MAC} \\
\quad [m_1^j \quad x \vdash_{pair} P_1] & \text{Create packet } P_1 \\
\quad B_1^j(P_1) & \text{Start to broadcast } P_1
\end{array}
$$

$$
\begin{array}{ll}
S_i^j(K_{i-1}^j, K_i^j, \ldots) \doteq & \\
\quad [m_i^j \quad K_i^j \vdash_{mac} x] & \text{Compute MAC} \\
\quad [m_i^j \quad x \vdash_{pair} P_i] & \text{Create packet } P_i \\
\quad B_i^j(P_i, K_{i-1}^j) & \text{Start to broadcast } P_i \text{ and disclose key } K_{i-1}
\end{array}
$$

$$
\begin{array}{rcll}
B_i^j(P_i) & \doteq & \overline{c_i}P_i.B_i^j(P_i) + tick.S_{i+1}^j(K_i^j, \ldots) & i = 0, 1 \\
B_i^j(P_i, K_{i-1}^j) & \doteq & \overline{c_i}P_i.\overline{c_i}K_{i-1}^j.B_i^j(P_i, K_{i-1}^j) + tick.S_{i+1}^j(K_i^j, \ldots) & i \geq 2
\end{array}
$$

Construct $B_i^j(\ldots)$ is responsible for potentially sending packets (and keys) an unbounded number of times, in order to simulate broadcast sessions. Sender S^j remains in the same state repeatedly sending messages unless the non-deterministic

[6] Actually, we consider constants with an arbitrary number of parameters. We could avoid this by considering, for modeling purposes, a special function fun, not available to possible adversaries, that may be used to represent the keys as a sequence.

[7] We remind the reader that the whole formalization we are going to give is based on personal choices since some details are not explicitly given in [PST+02]. In particular, the mechanism through which a receiver possibly identifies each sender process (and consequently each stream) is not defined in [PST+02], since the original construction is described with a single sender.

$$\frac{m \quad m'}{(m,m')}(\vdash_{pair}) \qquad \frac{(m,m')}{m}(\vdash_{fst}) \qquad \frac{(m,m')}{m'}(\vdash_{snd})$$

$$\frac{F \quad m}{F(m)}(\vdash_{one-way}) \qquad \frac{m \quad k}{mac(m,k)}(\vdash_{mac})$$

Figure 3.4: Inference system for μTESLA.

choice is resolved by choosing the derivative of the second summand in B_i^j; this causes a time unit to pass (a $tick$ action is performed). The construction models the behaviour of a wireless antenna making signals available only in a particular time interval. The presence of a non-deterministic choice in the construct makes it possible the passage to the following time interval without performing any (eventually zero) communication. This may implicitly model the unreliability of the wireless transmission and the occurrence of packet loss.

Among the receivers' set, each process behaves in the same way. The generic receiver process at step i is parameterized by a commitment to the key chain (let it be K_0^j) and by the packets it should still authenticate. We assume the receiver's set is divided into subgroups, each of them sharing a particular K_{SM} with one sender process. Sender S^j and receivers belonging to subgroup number j share K_{SM}^j. K_{SM}^j may denote a particular service each element in subgroup j is devoted to. Let us consider pay per view-based applications: among the receivers' set, the subgroup knowing K_{SM}^j may consist of all the paying spectators for movie number j. For environments closer to those depicted for μ-TESLA, let us consider a scenario in which sensors are used to periodically transmit readings regarding heating and air conditioning control in a building (and consequently receive broadcasted messages for routing updates or reprogramming): sensors in subgroup j may be all the sensors devoted to carry out the service for room number j. (S^j being the base station responsible for room number j.).

Below, we refer to $R_i^{j,q}$ to indicate the q-th receiver process belonging to sub-

group j and acting at step i.

$R_0^{j,q}(null) \doteq$	
$c_0(x).$	Receive first packet
$[x \vdash_{fst} x_{K_0}]$	Extract commitment to the key chain
$[x_{K_0} \quad K_{SM}^j \vdash_{mac} z]$	Compute MAC
$[x \vdash_{snd} x_{mac}]$	Extract MAC
$[z = x_{mac}]$	Verify MAC: if verified:
$tick.R_1^{j,q}(x_{K_0});$	Allow a time unit to pass and go to next state
$R_0^{j,q}(null)$	Wait for key

Upon receiving a value x on channel c_0, the receiver verifies the correctness of the commitment to the key chain, x_{K_0}: he computes $mac(x_{K_0}, K_{SM}^j)$ and he compares it with the message authentication code in the received packet. If the two MACs match, a time unit passes and the receiver goes to the next state, otherwise the receiver remains in the same state waiting for the right key K_{SM}^j.

Throughout the formalization, $null$ means an empty field.

$R_1^{j,q}(x_{K_0}) \doteq$	
$(c_1(y).$	Receive packet
$tick.R_2^{j,q}(y, x_{K_0})$	Allow a time unit to pass and go to next state
$) + tick.R_2^{j,q}(null, x_{K_0})$	Go to next state after a time unit

$R_1^{j,q}$ is willing to accept any arbitrary packet, because it cannot perform any verification yet. If nothing is received before the end of a time unit, transition takes place to next state $R_2^{j,q}$.

$R_i^{j,q}(p_{i-1}, x_{K_0}) \doteq$	
$c_i(p_i).R_i'^{j,q}(p_i, p_{i-1}, x_{K_0})$	Receive i-th packet; go to intermediary state $R_i'^{j,q}$
$+tick.R_{i+1}^{j,q}(null, x_{K_0})$	Go to next state after a time unit

$R_i^{j,q}$ is willing to accept packet P_i and travels to an intermediary state $R_i'^{j,q}$. If nothing is received before the end of a time unit, transition takes place to the next

state.

$$
\begin{array}{ll}
R_i'^{j,q}(p_i, p_{i-1}, x_{K_0}) \doteq & \\
\quad c_i(x_{K_{i-1}}). & \text{Receive key packet} \\
\quad [x_{K_0} = F^{i-1}(x_{K_{i-1}})] & \text{Verify the key w.r.t. the commitment} \\
\quad [p_{i-1} \vdash_{fst} y_{pay}] & \text{Extract payload} \\
\quad ([y_{pay} \quad x_{K_{i-1}} \vdash_{mac} z] & \text{If } x_{K_{i-1}} = K_{i-1}^j \text{ then: Compute MAC} \\
\qquad [p_{i-1} \vdash_{snd} y_{mac}] & \text{Extract MAC} \\
\qquad [z = y_{mac}] & \text{Verify MAC} \\
\qquad \overline{app} y_{pay}. & \text{Send } m_1^j \text{ to application level} \\
\qquad tick.R_{i+1}^{j,q}(p_i, x_{K_0}) & \text{Allow a time unit to pass and go to next state} \\
\quad); R_i'^{j,q}(p_i, p_{i-1}, x_{K_0}) & \text{Wait for key}
\end{array}
$$

In intermediary state $R_i'^{j,q}$ receives a key packet and verifies the correctness of the key w.r.t. the authenticated commitment $x_{K_0} = K_0^j$. Given the collision-free property of one-way functions, if the verification does not succeed it means $x_{K_{i-1}} \neq K_{i-1}^j$ and $R_i'^{j,q}$ simply stays in the same state waiting for the right subgroup key. If the verification succeeds, the correctness of P_{i-1} is verified by checking that the enclosed MAC is authentic. The successful outcome is here modeled by a scenario where the receiver sends the payload of the accepted packet over channel app[8].

Suppose packet P_{i-1} was correctly received, suppose also packet disclosing K_{i-1}^j is lost. At step i the receiver still cannot authenticate packet P_{i-1}. The key chain mechanism of the original protocol takes into account such a possibility: in interval $i+1$ the base station broadcasts key K_i^j, which the receiver authenticates by verifying $K_0^j = F^i(K_i^j)$. The receiver can authenticate P_i and derives $K_{i-1}^j = F(K_i^j)$, so it can also authenticate P_{i-1}. Actually, our formalization does not take into account recovering lost keys. For the sake of simplicity, we prefer to suppose that the key packet related to subgroup j is received (state $R_i'^{j,q}$).

We report below the formalization at step i, with $i \geq 2$, when a packet was not received at step $i-1$.

$$
\begin{array}{ll}
R_i^{j,q}(null, x_{K_0}) \doteq & \\
\quad c_i(p_i).tick.R_{i+1}^{j,q}(p_i, x_{K_0}) & \text{Receive i-th packet; go to next state} \\
\quad +tick.R_{i+1}^{j,q}(null, x_{K_0}) & \text{Go to next state after a time unit}
\end{array}
$$

[8]We omitted to insert an idling behavior when a deduction construct fails to be executed and in our formalization the system simply stops without letting time pass. This is not realistic, but it has no consequences since we use trace semantics for the analysis and makes it simpler.

3.3 Stable processes and compositional results for the non-timed setting

In this section a result about conditions for safe composition of processes will be given. In order to achieve this result, we should introduce the concept of stability of a process ([GLM03, GMPV03a, MPV03]).

Definition 16 *We say that a process P is stable w.r.t. ϕ_X, whenever for every X with $ID(X) \subseteq \phi_X$, $(P||X_{\phi_X}) \setminus C \stackrel{\gamma}{\Longrightarrow} (P'||X'_{\phi'_X}) \setminus C$ then $\mathcal{D}(\phi_X) = \mathcal{D}(\phi'_X)$.*

Basically, a process P is stable when an enemy with a certain knowledge ϕ_X does not increase significantly ϕ_X during the execution of P.

A series of lemmas are hereafter introduced. Their proofs are useful for the main result of this section, Proposition 1, introduced and proved at the end of the section.

Lemma 1 *Assume that P_1 and P_2 are stable w.r.t. ϕ_X; then $P_1||P_2$ is stable w.r.t. ϕ_X.*

Proof. For the sake of simplicity, we assume that $\phi_X = \mathcal{D}(\phi_X)$. By contradiction, consider a computation γ such that

$$(P_1||P_2||X_{\phi_X})\backslash C \stackrel{\gamma}{\Longrightarrow} (P_1''||P_2''||X_{\phi'_X})\backslash C$$

and $\mathcal{D}(\phi'_X) \neq \mathcal{D}(\phi_X) = \phi_X$. Then, it means that $P_1||P_2$ outputs at least one message m such that $m \notin \phi_X$. Let m_1 be the first (and last) of such messages sent during the computation under consideration. Let us assume, without loss of generality, that m_1 has been sent by P_1. Thus, we must have:

$$(P_1||P_2||X_{\phi_X})\backslash C \stackrel{\gamma'}{\Longrightarrow} (P_1'||P_2'||X_{\phi_X})\backslash C \stackrel{c!m_1}{\longrightarrow} (P_1''||P_2''||X_{\phi'_X})\backslash C$$

During the prefix of the computation, P_2 emitted over channels in C only messages in ϕ_X, since P_2 is stable by the hypothesis. Thus, that trace must be simulated by $P_1||X_{\phi_X}$. On the other hand, this implies: $(P_1||X_{\phi_X})\backslash C \stackrel{\gamma}{\Longrightarrow} (P_1''||X_{\phi'_X})\backslash C$ and $\mathcal{D}(\phi'_X) \neq \mathcal{D}(\phi_X)$. This is contradictory by the hypothesis.

Also, it is possible to prove the stability of the parallel composition of an arbitrary number n of processes, by induction on n. □

In the following, we will often exploit the fact that the general way in which Top_C^{ϕ} is specified implies that its behaviour includes that of any X belonging to the set $\mathcal{E}_C^{\phi_X}$ (proved in [FGM00a, FM99]).

Lemma 2 *Let P_1 and P_2 be stable processes w.r.t. ϕ; then,*

$$(P_1||P_2||Top_\phi^C) \setminus C \leq_{trace} (P_1||Top_\phi^C) \setminus C \;||\; (P_2||Top_\phi^C) \setminus C$$

Proof. The proof consists in checking that the following relation is a weak simulation:

$$\begin{aligned}\mathcal{R} \;=\; & \{((P'||P''||Top_\phi^C) \setminus C, (P'||Top_\phi^C) \setminus C \;||\; (P''||Top_\phi^C) \setminus C) \mid \\ & (P_1||P_2||Top_\phi^C) \setminus C \stackrel{\gamma}{\Longrightarrow} (P'||P''||Top_\phi^C) \setminus C, \#^{tick}(\gamma) = i\}\end{aligned}$$

- Assume $(P'||P''||Top_\phi^C) \setminus C \stackrel{a!m}{\longrightarrow} (P_1'||P''||Top_\phi^C) \setminus C$ due to a transition $P' \stackrel{a!m}{\longrightarrow} P_1'$, with $a \notin C$. Then, also $(P'||Top_\phi^C) \setminus C \;||\; (P''||Top_\phi^C) \setminus C \stackrel{a!m}{\longrightarrow} (P_1'||Top_\phi^C) \setminus C||(P''||Top_\phi^C) \setminus C$.

  A similar reasoning holds even if the transition is due to a receiving action by P' or when a τ action is performed by one of the two processes.

- Assume $(P'||P''||Top_\phi^C) \setminus C \stackrel{\tau}{\longrightarrow} (P_1'||P_1''||Top_\phi^C) \setminus C$ due to a transition $P' \stackrel{a!m}{\longrightarrow} P_1'$ and $P'' \stackrel{a(m)}{\longrightarrow} P_1''$. If $a \notin C$ then the case is trivial. If $a \in C$, since $m \in \mathcal{D}(\phi)$ ($P_1'||P_1''$ is a stable process), we have that $(P'||Top_\phi^C) \setminus C \stackrel{\tau}{\longrightarrow} (P_1'||Top_\phi^C) \setminus C$ and $(P''||Top_\phi^C) \setminus C \stackrel{\tau}{\longrightarrow} (P_2''||Top_\phi^C) \setminus C$. Thus, we get

  $$\begin{array}{ll}(P'||Top_\phi^C) \setminus C \;||\; (P''||Top_\phi^C) \setminus C & \stackrel{\tau}{\longrightarrow} \\ (P_1' \;||\; Top_\phi^C) \setminus C \;||\; (P''||Top_\phi^C) \setminus C & \stackrel{\tau}{\longrightarrow} \\ (P_1'||Top_\phi^C) \setminus C \;||\; (P_1''||Top_\phi^C) \setminus C & \end{array}$$

  The case of a synchronization between P' (P'') and Top_ϕ^C is a simplified instance of this case.

□

We can generalize the previous result as follows.

Lemma 3 *Let $\{P_j\}_{j=1,\ldots,n}$ be stable processes w.r.t. ϕ; then,*

$$(P_1|| \ldots ||P_n||Top_\phi^C) \setminus C \leq_{trace} (P_1||Top_\phi^C) \setminus C \;||\; \ldots \;||\; (P_n||Top_\phi^C) \setminus C$$

Proof. By induction on n. The base case is trivial. The case $n+1$ may be treated as follows:

$$\begin{array}{lll} (P_1||\ldots||P_n||P_{n+1}||Top_\phi^C) \setminus C & \leq_{trace} & \text{Lemma 2} \\ (P_1||\ldots||P_n||Top_\phi^C) \setminus C||(P_{n+1}||Top_\phi^C) \setminus C & \leq_{trace} & \text{Induction hypothesis} \\ (P_1||Top_\phi^C) \setminus C||\ldots||(P_{n+1}||Top_\phi^C) \setminus C & & \end{array}$$

□

Thus, the following compositional rule holds for the $GNDC^{\alpha}_{\leq_{trace}}$ schema (under the assumption that the involved processes are stable).

Proposition 1 *Given ϕ and a set of public channels C, assume processes $P_r \in GNDC^{\alpha_r(P_r)}_{\leq_{trace}}$ with $1 \leq r \leq n$ and P_r stable w.r.t. ϕ. It follows that $(P_1||\ldots||P_n)$ is stable w.r.t. ϕ and $(P_1||\ldots||P_n) \in GNDC^{\alpha_1(P_1)||\ldots||\alpha_n(P_n)}_{\leq_{trace}}$.*

Proof.

$$\begin{array}{lll} (P_1||P_2||\ldots||P_n||Top_\phi^C) \setminus C & \leq_{trace} & \text{Lemma 3} \\ (P_1||Top_\phi^C) \setminus C||\ldots||(P_n||Top_\phi^C) \setminus C & \leq_{trace} & (\leq_{trace} \text{ is a pre-congruence w.r.t. } ||) \\ \alpha_1(P_1)||\ldots||\alpha_n(P_n) & & \end{array}$$

Note 1 *A behavioral relation $\lhd : \mathcal{P} \rightarrow \mathcal{P}$ between processes is a pre-congruence with respect to $||$ if it is a pre-order and if, for every $P, Q, Q' \in \mathcal{P}$, if $Q \lhd Q'$ then $P||Q \lhd P||Q'$. In particular, the fact that $\leq_{trace}$ is pre-congruence with respect to the parallel operator has been proved in [FM99].*

□

3.4 Time-dependent stable processes and compositional results for the timed setting

In this section a result about conditions for safe composition of timed processes will be given. In order to achieve this result, we refine the concept of stability defined in [GLM03] and reminded in the previous section. To fix this definition, we briefly recall that the simple stability basically requires that the intruder knowledge does not increase when composing the intruder process with a process P. If so, we call P a stable process. As already discussed, (again, see Section 3.3 and [GLM03]), if we assume that the intruder knowledge does not increase when

composing the intruder process with P (*i.e.*, $P||X$) and with Q (*i.e.*, $Q||X$) (using the same communication channels) then the intruder knowledge does not increase when composing the intruder itself with the process $P||Q$. Unfortunately, such a form of stability is not time-dependent, *i.e.*, it takes into account the same knowledge during all the temporal execution of the processes at stake. This does not make it feasible to check properties based on a timed notion of secrecy and, consequently, to check protocols as μTESLA, whose security features exactly depend on a form of timed secrecy. We give now a refined notion of stability, called time-dependent stability, that allows us to cope with timed secrecy and so also with security properties of protocols that rely on it.

We let γ be a sequence of actions (possibly empty) ranging over $Act\backslash\{\tau\}$. Let $\#^{tick}(\gamma)$ be the number of occurrences of $tick$ actions in the sequence γ.

Definition 17 *Let X_ϕ be the closed term in X belonging to the messages deducible from ϕ. We say that a process P is time-dependent stable w.r.t. the sequence $\{\phi_i\}_{i\geq 0}$ if, whenever $(P||X_{\phi_0})\backslash C \stackrel{\gamma}{\Longrightarrow} (P'||X'_{\phi'})\backslash C$ and $\#^{tick}(\gamma) = i$, then $\mathcal{D}(\phi') = \mathcal{D}(\phi_i)$.*

Basically, a process P is time-dependent stable if an enemy cannot increase significantly its knowledge when P runs in the space of a time slot.

A series of lemmas are hereafter introduced. Their proofs are useful for the main result of this section, Proposition 2, introduced and proved at the end of the section. The proofs are somehow similar to the ones shown in the previous sections. They are however introduced for the sake of completeness.

Lemma 4 *Assume that P_1 and P_2 are t.d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$; then $P_1||P_2$ is t.d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$.*

Proof. For the sake of simplicity, we assume that $\phi_i = \mathcal{D}(\phi_i), \forall i$. By contradiction, consider a computation γ such that

$$(P_1||P_2||X_{\phi_0})\backslash C \stackrel{\gamma}{\Longrightarrow} (P''_1||P''_2||X_{\phi''})\backslash C$$

and $\mathcal{D}(\phi'') \neq \mathcal{D}(\phi_i) = \phi_i$. Then, it means that $P_1||P_2$ outputs at least one message m s.t. $m \notin \phi_i$. Let m_1 be the first (and last) of such messages sent during the computation under consideration. Assume, without loss of generality, that m_1 has been sent by P_1. Thus, we must have:

$$(P_1||P_2||X_{\phi_0})\backslash C \stackrel{\gamma'}{\Longrightarrow} (P'_1||P'_2||X_{\phi_i})\backslash C \stackrel{\overline{c}m_1}{\longrightarrow} (P''_1||P''_2||X_{\phi''})\backslash C$$

During the prefix of the computation, P_2 emitted over channels in C only messages in ϕ_i, for each time slot i, since P_2 is t.d. stable by hypothesis. Thus, that trace must be simulated by $P_1||X_{\phi_i}$. This would imply: $(P_1||X_{\phi_0})\backslash C \stackrel{\gamma}{\Longrightarrow} (P_1''||X_{\phi''})\backslash C$ and $\mathcal{D}(\phi'') \neq \mathcal{D}(\phi_i)$. This cannot be possible since P_1 is t.d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$.

The same result holds for an arbitrary number n of processes (proved by induction on n). □

Lemma 5 *Assume that P_1 and P_2 are t.d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$; then,*

$$(P_1||P_2||tTop^C_{\phi_0}) \setminus C \leq_{ttrace} (P_1||tTop^C_{\phi_0}) \setminus C||(P_2||tTop_{\phi_0}) \setminus C$$

Proof. The proof consists in checking that the following relation is a weak simulation:

$$\begin{aligned}\mathcal{R} \;=\; & \{((P'||P''||tTop^C_{\phi_i}) \setminus C, (P'||tTop^C_{\phi_i}) \setminus C||(P''||tTop^C_{\phi_i}) \setminus C) \mid \\ & (P_1||P_2||tTop^C_{\phi_i}) \setminus C \stackrel{\gamma}{\Longrightarrow} (P'||P''||tTop^C_{\phi_i}) \setminus C, \#^{tick}(\gamma) = i\}\end{aligned}$$

- Assume $(P'||P''||tTop^C_{\phi_i}) \setminus C \stackrel{\overline{a}m}{\longrightarrow} (P_1'||P''||tTop^C_{\phi_i}) \setminus C$ due to a transition $P' \stackrel{\overline{a}m}{\longrightarrow} P_1'$, with $a \notin C$. Then, also $(P'||tTop^C_{\phi_i}) \setminus C||(P''||tTop^C_{\phi_i}) \setminus C \stackrel{\overline{a}m}{\longrightarrow} (P_1'||tTop^C_{\phi_i}) \setminus C||(P''||tTop^C_{\phi_i}) \setminus C$.

  A similar reasoning holds when the transition is due to a receiving action by P' or one of the two processes performs an internal computation.

- Assume $(P'||P''||tTop^C_{\phi_i}) \setminus C \stackrel{\tau}{\longrightarrow} (P_1'||P_1''||tTop^C_{\phi_i}) \setminus C$ due to a transition $P' \stackrel{\overline{a}m}{\longrightarrow} P_1'$ and $P'' \stackrel{a(m)}{\longrightarrow} P_1''$. The case $a \notin C$ is trivial. If $a \in C$, since $m \in \mathcal{D}(\phi_i)$ ($P_1'||P_1''$ is t.d. stable), we have that $(P'||tTop^C_{\phi_i}) \setminus C \stackrel{\tau}{\longrightarrow} (P_1'||tTop^C_{\phi_i}) \setminus C$ and $(P''||tTop^C_{\phi_i}) \setminus C \stackrel{\tau}{\longrightarrow} (P_2''||tTop^C_{\phi_i}) \setminus C$. Thus, we get

$$\begin{array}{l}(P'||Top_{\phi_i}) \setminus C||(P''||Top_{\phi_i}) \setminus C \;\stackrel{\tau}{\longrightarrow} \\ (P_1'||Top_{\phi_i}) \setminus C||(P''||Top_{\phi_i}) \setminus C \;\stackrel{\tau}{\longrightarrow} \\ (P_1'||Top_{\phi_i}) \setminus C||(P_1''||Top_{\phi_i}) \setminus C\end{array}$$

  The case of a synchronization between P' (P'') and $tTop$ is a simplified instance of this previous case.

- Assume that $(P'||P''||tTop^C_{\phi_i}) \setminus C \xrightarrow{tick} (P'_1||P''_1||tTop^C_{\phi_{i+1}}) \setminus C$ is due to a time synchronization, *i.e.*, $P' \xrightarrow{tick} P'_1$, $P'' \xrightarrow{tick} P''_1$ and $tTop^C_{\phi_i} \xrightarrow{tick} tTop^C_{\phi_{i+1}}$. Then, $(P'||tTop^C_{\phi_i})\setminus C \xrightarrow{tick} (P'_1||tTop^C_{\phi_{i+1}})\setminus C$ and $(P''||tTop^C_{\phi_i})\setminus C \xrightarrow{tick} (P''_1||tTop^C_{\phi_{i+1}})\setminus C$ (due to time stability of P_1 and P_2 and henceforth of P' and P''). Thus, we get

$$(P'||tTop^C_{\phi_i})\setminus C \;||\; (P''||tTop^C_{\phi_i})\setminus C \xrightarrow{tick} (P'_1||tTop^C_{\phi_{i+1}})\setminus C \;||\; (P''_1||tTop^C_{\phi_{i+1}})\setminus C$$

□

We can generalize the previous result as follows.

Lemma 6 *Assume that $\{P_j\}_{j=1,\ldots,n}$ are t.d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$; then,*

$$(P_1||\ldots||P_n||tTop^C_{\phi_0}) \setminus C \leq_{ttrace} (P_1||tTop^C_{\phi_0}) \setminus C||\ldots||(P_n||tTop^C_{\phi_0}) \setminus C$$

Proof. By induction on n. The base case is trivial. The case $n+1$ may be treated as follows:

$$\begin{array}{lll} (P_1||\ldots||P_n||P_{n+1}||tTop^C_{\phi}) \setminus C & \leq_{trace} & \text{Lemma } 5 \\ (P_1||\ldots||P_n||tTop^C_{\phi}) \setminus C||(P_{n+1}||tTop^C_{\phi}) \setminus C & \leq_{trace} & \text{Induction hypothesis} \\ (P_1||Top^C_{\phi}) \setminus C||\ldots||(P_{n+1}||tTop^C_{\phi}) \setminus C & & \end{array}$$

□

Proposition 2 *Given a sequence $\{\phi_i\}_{i\geq 0}$ and a set of public channels C, assume $P_r \in tGNDC^{\alpha_r(P_r)}_{\leq_{ttrace}}$ with $1 \leq r \leq n$. Assume also P_r t. d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$. It follows that $(P_1||P_2||\ldots||P_n) \in tGNDC^{\alpha_1(P_1)||\alpha_2(P_2)||\ldots||\alpha_n(P_n)}_{\leq_{ttrace}}$ and $(P_1||P_2||\ldots||P_n)$ is t. d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$.*

Proof.

$$\begin{array}{lll} (P_1||P_2||\ldots||P_n||Top_{\phi_0}) \setminus C & \leq_{ttrace} & \text{Lemma } 6 \\ (P_1||Top_{\phi_0}) \setminus C||\ldots||(P_n||Top_{\phi_0}) \setminus C & \leq_{ttrace} & (\leq_{ttrace} \text{ is a pre-congruence w.r.t. } ||) \\ \alpha_1(P_1)||\ldots||\alpha_n(P_n) & & \end{array}$$

The fact that $\leq_{ttrace}$ is a pre-congruence w.r.t. $||$ has been proved in [GLM03] and reminded in [GMPV03b]. □

Example 2 *The process $P = tick.\overline{c}k.\mathbf{0}'$ enjoys the secrecy of k for one time unit. In the more complex process $Q = (c(x).[x = k]\overline{c}m) + tick.\mathbf{0}'$ the secrecy of k in the first time unit is crucial to get the secrecy of m. Indeed, either Q is willing to receive the key k only in the first time unit (if so, it releases m) or it starts to idle. We have that P and Q are t.d. stable w.r.t. $\phi_0 = \{\emptyset\}$, $\phi_i = \mathcal{D}(\{k\})$ for $i \geq 1$. Then, $P||Q$ is t.d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$ (by Proposition 2) and so m will never belong to the knowledge of the intruder (whose initial knowledge is $\emptyset$).*

3.5 An analysis of the EMSS protocol: integrity

Proposition 1 will be applied in this section for verifying integrity of the (1,2) EMSS. The specification of the protocol has been given in Subsection 3.2.2.

A more general specification, with an arbitrary number n of receivers, is $S_0||\overbrace{Rec_0||\ldots||Rec_0}^{n}$.

Integrity is defined within the GNDC schema as the ability to accept only the message m_i by a receiver as the $i-th$ message sent by the sender (assuming m_i is not lost). Let us assume that a receiver signals the acceptance of a stream of messages as a legitimate one, by issuing it, as a unique list of messages, on a special channel c_{app}. Thus, let α_{int} be $Spec_{sign} = \sum_{s\in streams} c_{app}!s.\mathbf{0}$, where $streams$ is the set of all the possible ordered sub streams of $m_0 \ldots m_{last}$.

Definition 18 *A system P, consisting of a sender of a stream of messages $\{m_i\}$ and a receiver, enjoys the integrity property whenever $P \in GNDC^{\alpha_{int}}_{\leq_{trace}}$.*

Basically, integrity holds when the receiver accepts exactly a subset of the messages m_i in the correct order even in presence of an adversary. The key point is that the intruder will never acquire the private key of the sender to successfully sign the final packet of the stream.

In a multi-receiver setting with one sender, a protocol guarantees integrity whenever each receiver accepts only the stream of messages that the sender wishes to deliver. In our case, the specification for n receivers is simply the parallel composition of α_{int} n-times.

S_0, Rec_0 are stable w.r.t. the following initial knowledge ϕ_X:

$$\phi_X = \{P_0\} \cup \{P_1\} \cup \{P_i \mid i = 2, \ldots, last\} \cup \{pk(S), P_{sign}\}$$

This can be verified by looking at the specifications of S_0 and Rec_0.

The initial knowledge ϕ_X includes indeed all the messages an adversary would be able to add to its knowledge by eavesdropping on a run of the protocol (in other words, X does not increase its knowledge when S_0 and Rec_0 run.

This implies that the considered intruder has the most powerful means to act since the beginning of the computation. One may comment that this is not correct, since it does not follow the reality. On the other hand, this is only a trick in the model, and, if the protocol satisfies the integrity property in this very hostile environment, then it means that it will satisfy this property in a less powerful one. This may be formally justified, [FM99]. Here, we prefer to give an informal discussion of the matter: let us suppose that there exists a sequence of actions, leading to an attack w.r.t. a procedure, performed by an intruder whose initial knowledge is ϕ. Then, let us suppose that the intruder knows ϕ', with $\phi \in \phi'$. Again, there will be at least the attack found starting from ϕ. On the other hand, if no attack exists with ϕ', one may reasonably conclude that no attack will exist by starting from a subset ϕ of ϕ'.

We are able to prove that S_0 enjoys $GNDC^{\mathbf{0}}_{\leq_{trace}}$ and Rec_0 enjoys $GNDC^{\alpha_{int}}_{\leq_{trace}}$, that is to say for all $X \in \mathcal{E}_C^{\phi_X}$ we have $(S_0||X)\backslash C \leq_{trace} \mathbf{0}$ and $(Rec_0||X)\backslash C \leq_{trace} \alpha_{int}$. This may be done by finding a suitable weak simulation relation between $(S_0||X) \setminus C$ and $\mathbf{0}$, and between $(Rec_0||X) \setminus C$ and $Spec_{sign}$ ($\forall X \in \mathcal{E}_C^{\phi_X}$), respectively. (The easier way is to prove the same with one check, by simply considering the top element Top_ϕ^C).

Let $C = \{c_{sign}\} \cup \{c_i \mid 0 \leq i \leq last\}$ be the set of channels over which each element of set $\mathcal{E}_C^{\phi_X}$ is able to communicate.

The candidate weak simulation relation we consider for dealing with the sender specifications is the following:

$$\begin{aligned}\mathcal{R}_{\mathcal{S}} = \quad & (((S_i(...)||X)\backslash C, \mathbf{0}) \mid X \in \mathcal{E}_C^{\phi_X}, 0 \leq i \leq last) \\ & \cup(((S_{sign}(...)||X)\backslash C, \mathbf{0}) \mid X \in \mathcal{E}_C^{\phi_X})\end{aligned}$$

The candidate weak simulation relation we consider for dealing with the re-

ceiver specifications is the following:

$$\begin{aligned}
\mathcal{R}_{\mathcal{R}} = \; & (((Rec_0(0,0,0)||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}) \\
& \cup(((Rec_1(0,0,0)||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}) \\
& \cup(((Rec_1(x_{P_0},0,0)||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}) \\
& \cup(((Rec_i(x_{P_{j1}}, x_{P_{j2}}, tup_{\{m_j\}}^{i-1})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}, 2 \leq i \leq last) \\
& \cup(((Rec'_i(x_{P_i}, x_{P_{i-1}}, tup_{\{m_j\}}^{i-1})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}, 2 \leq i \leq last) \\
& \cup(((Rec''_i(x_{P_i}, x_{P_{i-2}}, tup_{\{m_j\}}^{i-1})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}, 2 \leq i \leq last) \\
& \cup(((Rec_{sign}(x_{P_{j1}}, x_{P_{j2}}, tup_{\{m_j\}}^{last})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}) \\
& \cup(((Rec^*_{sign}(x_{P_{sign}}, x_{P_{j1}}, x_{P_{j2}}, tup_{\{m_j\}}^{last})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}) \\
& \cup(((Rec'_{sign}(x_{ver}, x_{P_{last}}, tup_{\{m_j\}}^{last})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}) \\
& \cup(((Rec''_{sign}(x_{ver}, x_{P_{last-1}}, tup_{\{m_j\}}^{last})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})
\end{aligned}$$

$tup_{\{m_j\}}^{i-1}, tup_{\{m_j\}}^{last}$ are lists of meaningful payloads (also updated). By inspection of the possible cases we may show that $\mathcal{R}_{\mathcal{S}}$ and $\mathcal{R}_{\mathcal{R}}$ are weak simulations. We omitted to explicitly put in $\mathcal{R}_{\mathcal{S}}$ and $\mathcal{R}_{\mathcal{R}}$ the pairs in which the first process performs deduction constructs.

The reader will probably be bored with the long pen-and-paper proofs required to prove that those pairs form a weak simulation. Here, we give only a sketch of the proof dealing with the receiver specification. Nevertheless, the interested reader will find a very similar proof, in its completeness, in the next section.

When the first process performs inference (or match) constructs and it gets stuck because an inference rule does not apply, or simply travels to the next state, it can be weakly simulated by whatever process, in particular $Spec_{sign}$. When Rec_0 performs a receiving action, the process on the left may perform a τ action and it can be weakly simulated by whatever process, in particular $Spec_{sign}$. The interesting case is when the first process outputs a tuple of messages $tup_{\{m_j\}}$ over channel $c_{app} \notin C$. In this case, it must be $\{x_{ver}\}_{sk(S)} = P_{sign}$ and, assuming that digital signatures and hash functions cannot be forged, all the messages in $tup_{\{m_j\}}$ must be replaced with one of all the possible ordered sub streams of $m_0 \ldots m_{last}$. This can be weakly simulated by $Spec_{sign}$ that has been defined as the process sending all the possible ordered sub streams of $m_0 \ldots m_{last}$.

Each resulting pair consisting of the derivatives still belong to $\mathcal{R}_{\mathcal{R}}$.

Proposition 3 *$S_0 \in GNDC^{\mathbf{0}}_{\leq trace}$ and $Rec_0 \in GNDC^{\alpha_{int}}_{\leq trace}$.*

The following proposition follows by the fact that S_0, Rec_0 are stable w.r.t. ϕ_X, by Proposition 1 and Proposition 3.

Proposition 4 $S_0||Rec_0 \in GNDC^{\alpha_{int}}_{\leq trace}$.

Then, the following statement holds because Proposition 1 is applicable once again.

Proposition 5 *The (1,2) EMSS Protocol enjoys integrity for whatever number of receivers.*

To check a systems with an arbitrary number of components, what we did is simply consider the components separately. The result follows by Proposition 1 where index r is not fixed *a priori* and $P_1 = S_0$ and $P_r, 2 \leq r \leq n$ is Rec_0.

In [GMPV03a], compositional principles have been applied also to the Gennaro-Rohatgi scheme (in particular, that paper reports a detailed analysis of the off-line case, whereas the proofs for the on-line case are sketched.). In both the cases, the stability condition and the fulfillment of GNDC of a sender and an unbounded number of receivers with respect to a certain fixed intruder initial knowledge are proved.

3.6 An analysis of the μTESLA protocol: timed integrity

Proposition 2 will be applied in this section for verifying timed integrity of the μTESLA protocol. The specification of the protocol has been given in Subsection 3.2.3.

So called timed integrity belongs to a new class of security properties defined in [GLM03]. A stream signature protocol guarantees timed integrity on a set of messages $\{m_i\}$ if, whenever the generic receiver accepts an item in a time interval i, let us say item x, then $x = m_{i-\delta}$, $i - \delta$ being the time interval in which x has been received. ($\delta = 1$ in the formalization of μTESLA given in Subsection 3.2.3.)

In μTESLA, let us assume that a receiver signals the acceptance of a payload as a legitimate one, by issuing it on a special channel app.

Let $P^q \doteq S^j_0||R^{j,q}_0$ be the system consisting of a single sender and the q-th receiver in subgroup j, sharing K^j_{SM}. Let function $\alpha_{tInt}(P^q)$ be $tSpec_0$ where

$$
\begin{aligned}
&tSpec_0 \doteq tick.tSpec_1 \\
&tSpec_1 \doteq tick.tSpec_2 \\
&tSpec_i \doteq tick.tSpec_{i+1} + \overline{app}(m^j_{i-1}).tick.tSpec_{i+1} \quad i \geq 2
\end{aligned}
$$

$\alpha_{tInt}(P^q)$ may denote the correct external behaviour of P^q. In the first two steps it simply let time pass, while in further steps it may either let time pass (denoting packet loss) or let a verified payload to be sent on the special channel app and then let time pass. The set of all messages sent on channel app is the set of all the possible ordered substreams of $\{m_i^j\}_{i\geq 1}$. Let function $\alpha_{tInt}^j(P^j) \doteq \Pi_{1\leq q\leq n_j}\alpha_{tInt}(P^q)$, n_j being the cardinality of the receivers in subgroup j.

Definition 19 *The system $P^j \doteq S_0^j||R_0^{j,1}||R_0^{j,2}||\ldots||R_0^{j,n_j}$, consisting of a sender of streamed data $\{m_i^j\}$ and the receivers in subgroup j enjoys the timed integrity property whenever $P^j \in tGNDC_{\leq_{ttrace}}^{\alpha_{tInt}^j(P^j)}$.*

Basically, it means that each receiver accepts exactly the messages belonging to $\{m_i^j\}$ in the correct order and within the time interval following the one in which the sender actually sent the messages, even in presence of an intruder (unless packets P_i are lost). The key point is that the intruder will never acquire the shared key K_{SM}^j to establish a secure channel over which the commitment to the key chain is exchanged[9].

We first consider system P^q. We may prove that S_0^j and $R_0^{j,q}$ (Subsection 3.2.3) are t.d. stable w.r.t. the sequence $\{\phi_i\} = \phi_0, \phi_1, \phi_2, \ldots$ defined as follows:

$$\begin{aligned}
&\phi_0 = \{K_0^j, mac(K_0^j, K_{SM}^j) \mid 1 \leq j \leq n\}\\
&\phi_1 = \phi_0 \cup \{m_1^j, mac(m_1^j, K_1^j) \mid 1 \leq j \leq n\}\\
&\phi_2 = \phi_1 \cup \{m_2^j, mac(m_2^j, K_2^j), K_1^j \mid 1 \leq j \leq n\}\\
&\ldots\\
&\phi_i = \phi_{i-1} \cup \{m_i^j, mac(m_i^j, K_i^j), K_{i-1}^j \mid 1 \leq j \leq n\}\\
&\ldots
\end{aligned}$$

where n is the number of senders. ϕ_i is equal to ϕ_{i-1} plus the set of all the messages an intruder would be able to add to its knowledge by eavesdropping on a run of the protocol during the whole time interval i (of course including those messages coming from all the other senders processes). The same considerations about the power of the intruder hold as in the previous section. Actually, the intruder have more powerful means to act since the beginning of each time interval.

We are going to prove that S_0^j enjoys $tGNDC_{\leq_{ttrace}}^{\mathbf{0}'}$ and $R_0^{j,q}$ enjoys $tGNDC_{\leq_{ttrace}}^{\alpha_{tInt}(P^q)}$, that is to say for all $X \in t\mathcal{E}_C^{\phi_0}$ we have $(S_0^j||X)\backslash C \leq_{ttrace} \mathbf{0}'$ and $(R_0^{j,q}||X)\backslash C \leq_{ttrace} \alpha_{tInt}(P^q)$. This may be done by finding a suitable weak

[9]We remind the reader that $K_{SM}^m \neq K_{SM}^n$ if $m \neq n$ and $K_i^m \neq K_l^n$ if $m \neq n$ or $i \neq l$.

simulation relation between $(S_0^j||X_{\phi_0}) \setminus C$ and $\mathbf{0}'$ and between $(R_0^{j,q}||X_{\phi_0}) \setminus C$ and $tSpec_0$, respectively. The set C of channels over which an intruder is able to communicate is $C = \{c_i \mid i \geq 0\}$. The weak simulation relation dealing with the sender specifications is the following:

$$\begin{aligned}\mathcal{R}_{\mathcal{S}} \;=\; & (((S_i^j(...)||X_{\phi_i})\backslash C, \mathbf{0}') \mid \forall i, X_{\phi_i} \in t\mathcal{E}_C^{\phi_i}) \\ & \cup(((B_i^j(...)||X_{\phi_i})\backslash C, \mathbf{0}') \mid \forall i, X_{\phi_i} \in t\mathcal{E}_C^{\phi_i}) \\ & \cup(((\overline{c_i}K_{i-1}^j.B_i^j(\ldots)||X_{\phi_i})\backslash C, \mathbf{0}') \mid i > 1, X_{\phi_i} \in t\mathcal{E}_C^{\phi_i})\end{aligned}$$

The weak simulation relation we consider for dealing with the receiver specifications is the following (superscript q is omitted for simplicity):

$$\begin{aligned}\mathcal{R} = \; & (((R_0^j(null)||X_{\phi_0})\backslash C, tSpec_0) \mid X_{\phi_0} \in t\mathcal{E}_C^{\phi_0}) \\ & \cup((tick.(R_1^j(K_0^j)||X_{\phi_0})\backslash C, tSpec_0) \mid X_{\phi_0} \in t\mathcal{E}_C^{\phi_0}) \\ & \cup(((R_1^j(K_0^j)||X_{\phi_1})\backslash C, tSpec_1) \mid X_{\phi_1} \in t\mathcal{E}_C^{\phi_1}) \\ & \cup(((R_i^j(null, K_0^j)||X_{\phi_i})\backslash C, tSpec_i) \mid i \geq 2, X_{\phi_i} \in t\mathcal{E}_C^{\phi_i}) \\ & \cup((tick.(R_i^j(p_{i-1}, K_0^j)||X_{\phi_{i-1}})\backslash C, tSpec_{i-1}) \mid i \geq 2, X_{\phi_{i-1}} \in t\mathcal{E}_C^{\phi_{i-1}}) \\ & \cup(((R_i^j(p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i) \mid i \geq 2, X_{\phi_i} \in t\mathcal{E}_C^{\phi_i}) \\ & \cup(((R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i) \mid i \geq 2, X_{\phi_i} \in t\mathcal{E}_C^{\phi_i}) \\ & \cup(((R_i^j(x_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i) \mid fst(x_{i-1}) \neq m_{i-1}^j, i \geq 2, X_{\phi_i} \in t\mathcal{E}_C^{\phi_i}) \\ & \cup((tick.(R_i^j(x_{i-1}, K_0^j)||X_{\phi_{i-1}})\backslash C, tSpec_{i-1}) \mid fst(x_{i-1}) \neq m_{i-1}^j, i \geq 2, \\ & X_{\phi_{i-1}} \in t\mathcal{E}_C^{\phi_{i-1}}) \\ & \cup((tick.(R_i^j(p_{i-1}*, K_0^j)||X_{\phi_{i-1}})\backslash C, tick.tSpec_i) \mid i \geq 2, X_{\phi_{i-1}} \in t\mathcal{E}_C^{\phi_{i-1}})\end{aligned}$$

where $p_1, p_{i-1}, p_i*, p_{i-1}*$ and x_{i-1} are not empty fields. $p_i*, p_{i-1}*$ are shortcuts to denote either authentic packets sent by the sender or others. We omitted to explicitly put in $\mathcal{R}_{\mathcal{S}}$ and $\mathcal{R}$ the pairs in which the first process performs deduction constructs.

Lemma 7 *S_0^j and $R_0^{j,q}$ are t. d. stable w.r.t. $\{\phi_i\}$.*

Lemma 8 *$S_0^j \in tGNDC_{\leq_{ttrace}}^{\mathbf{0}'}$ and $R_0^{j,q} \in tGNDC_{\leq_{ttrace}}^{\alpha_{tInt}(P^q)}$*

Proof. Throughout the proof, we omit to consider the cases in which the sender and the receiver by themselves perform internal actions.

- $S_0^j \in tGNDC_{\leq_{ttrace}}^{\mathbf{0}'}$. Let us consider relation $\mathcal{R}_{\mathcal{S}}$. $\mathcal{R}_{\mathcal{S}}$ is a weak simulation:

- $((S_i^j||X_{\phi_i})\backslash C, \mathbf{0}')$. S_i^j may
  - either perform a $tick$ action: in this case the whole system on the left performs $tick$ and $(S_i^j||X_{\phi_i})\backslash C \xrightarrow{tick} (S_{i+1}^j||X_{\phi_{i+1}})\backslash C$. $\mathbf{0}'$ is able to simulate it and $((S_{i+1}^j||X_{\phi_{i+1}})\backslash C, \mathbf{0}') \in \mathcal{R}_{\mathcal{S}}$.
  - or go to intermediary state B_i^j. $\mathbf{0}'$ is able to simulate it and $((B_i^j||X_{\phi_i})\backslash C, \mathbf{0}') \in \mathcal{R}_{\mathcal{S}}$.
- $((B_i^j||X_{\phi_i})\backslash C, \mathbf{0}')$. B_i^j may perform a sending action, whereas X_{ϕ_i} synchronizes on that action: the whole system performs τ. It may happen:
  - $(B_i^j||X_{\phi_i})\backslash C \xrightarrow{\tau} (B_i^j||X_{\phi_i})\backslash C, i = 0, 1$. $\mathbf{0}'$ is able to simulate it and $((B_i^j||X_{\phi_i})\backslash C, \mathbf{0}') \in \mathcal{R}_{\mathcal{S}}$.
  - $(B_i^j||X_{\phi_i})\backslash C \xrightarrow{\tau} (\overline{c_i}K_{i-1}^j.B_i^j||X_{\phi_i})\backslash C, i \geq 1$. $\mathbf{0}'$ is able to simulate it and $((\overline{c_i}K_{i-1}^j.B_i^j||X_{\phi_i})\backslash C, \mathbf{0}') \in \mathcal{R}_{\mathcal{S}}$.
- $((\overline{c_i}K_{i-1}^j.B_i(\ldots)||X_{\phi_i})\backslash C, \mathbf{0}')$. The process on the left may perform a τ action, i.e. $(\overline{c_i}K_{i-1}^j.B_i^j(\ldots)||X_{\phi_i})\backslash C \xrightarrow{\tau} (B_i^j(\ldots)||X_{\phi_i})\backslash C$. Similar to the previous item.

- $R_0^{j,q} \in tGNDC_{\leq ttrace}^{\alpha_{tInt}(P^q)}$. Let us consider relation $\mathcal{R}$. $\mathcal{R}$ is a weak simulation:
  - $((R_0^j(null)||X_{\phi_0})\backslash C, tSpec_0)$. Suppose $R_0^j(null)$ performs a receiving action and X_{ϕ_0} the corresponding sending action. X_{ϕ_0} could have sent any message $\in \mathcal{D}(\phi_0)$ whereas the only message $R_0^j(null)$ will accept will be the MAC computed with key K_{SM}^j. In this case $(R_0^j(null)||X_{\phi_0})\backslash C \xrightarrow{\tau} tick.(R_1^j(K_0^j)||X_{\phi_0})\backslash C$ and $tSpec_0$ is able to simulate τ and $(tick.(R_1^j(K_0^j)||X_{\phi_0})\backslash C, tSpec_0) \in \mathcal{R}$. When the received message contains a MAC not computed with K_{SM}^j the system maintains the same configuration and $tSpec_0$ is able to simulate it.
  - $(tick.(R_1^j(K_0^j)||X_{\phi_0})\backslash C, tSpec_0)$. The first process may only perform $tick$ by reaching the configuration $(R_1^j(K_0^j)||X_{\phi_1})\backslash C$. Note that also $tSpec_0 \xrightarrow{tick} tSpec_1$ and $((R_1^j(K_0^j)||X_{\phi_1})\backslash C, tSpec_1) \in \mathcal{R}$.
  - $((R_1^j(K_0^j)||X_{\phi_1})\backslash C, tSpec_1)$.
    - The first process may perform $tick$ and go to $(R_2^j(null, K_0^j)||X_{\phi_2})\backslash C$. Note that also $tSpec_1 \xrightarrow{tick} tSpec_2$ and $((R_2^j(null, K_0^j)||X_{\phi_2})\backslash C, tSpec_2) \in \mathcal{R}$.

  - If $R_1^j(K_0^j)$ performs a receiving action and X_{ϕ_1} the corresponding sending action (by sending messages $\in \mathcal{D}(\phi_1)$), then $((R_1^j(K_0^j)||X_{\phi_1})\backslash C \xrightarrow{\tau} (tick.R_2^j(p_1*, K_0^j)||X_{\phi_1})$, where p_1* could be either the authentic packet send by the sender p_1 or another one x_1. Note that $((tick.R_2^j(p_1*, K_0^j)||X_{\phi_1}), tSpec_1) \in \mathcal{R}$.

- $(tick.(R_2^j(p_1, K_0^j)||X_{\phi_1})\backslash C, tSpec_1)$. The first process may only perform a tick action reaching the configuration $(R_2^j(p_1, K_0^j)||X_{\phi_2})\backslash C$. Note that also $tSpec_1 \xrightarrow{tick} tSpec_2$ and $((R_2^j(p_1, K_0^j)||X_{\phi_2})\backslash C, tSpec_2) \in \mathcal{R}$.

- $((R_i^j(p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$.

  - The first process may perform $tick$ by reaching $\{(R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C$. Note that also $tSpec_i \xrightarrow{tick} tSpec_{i+1}$ and $((R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C, tSpec_{i+1}) \in \mathcal{R}$.
  - If $R_i^j(p_{i-1}, K_0^j)$ performs a receiving action then $((R_i^j(p_{i-1}, K_0^j)||X_{\phi_i})\backslash C \xrightarrow{\tau} (R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C$. Note that $((R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i) \in \mathcal{R}$.

- $((R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$. If $R_i^{j'}$ outputs a message over channel app, it must be $z = y_{mac}$, $p_{i-1} = snd(y_{mac})$, $x_{K_{i-1}} = K_{i-1}^j$ and $fst(p_{i-1})$ must be replaced with m_{i-1}^j. $R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C \xrightarrow{\overline{app}m_{i-1}^j} tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i})\backslash C$ and $tSpec_i \xrightarrow{\overline{app}m_{i-1}^j} tick.tSpec_{i+1}$. Both the derivatives $\in \mathcal{R}$.

- $((tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i})\backslash C, tick.tSpec_{i+1})$. Both the processes may perform $tick$ and the derivatives $\in \mathcal{R}$.

- $((R_i^j(null, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$.

  - If $R_i^j(null, K_0^j)$ performs a receiving action and X_{ϕ_i} the corresponding sending action, then $((R_i^j(null, K_0^j)||X_{\phi_i})\backslash C \xrightarrow{\tau} (tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i})$. Note that $((tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i}), tSpec_i) \in \mathcal{R}$.
  - If the first process performs $tick$, it reaches the configuration $(R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C$. Note that also $tSpec_i \xrightarrow{tick} tSpec_{i+1}$ and $((R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C, tSpec_{i+1}) \in \mathcal{R}$

– $((R_i^j(x_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$. In this case the equality check among hashes does not succeed and the system gets stuck. $tSpec_i$ is always able to simulate it.

□

The following proposition follows by Lemmas 7 and 8 and by Proposition 2, where $r = 1, 2$, $P_1 = S_0^j$, $P_2 = R_0^{j,q}$.

Proposition 6 $P^q \in tGNDC^{\alpha_{tInt}(P^q)}_{\leq_{ttrace}}$[10].

The correctness of the multiple receivers version (considering all the receivers belonging to subgroup j), can be also proved using results of Lemmas 7 and 8 and Proposition 2, where index r is not fixed *a priori* and $P_1 = S_0^j$ and $P_r = R_0^{j,q}$ with $1 \leq q \leq n_j$.

Proposition 7 *System* P^j *(in Definition 19)* $\in tGNDC^{\alpha^j_{tInt}(P^j)}_{\leq_{ttrace}}$.

We get into the issue of considering a multiple senders/receivers environment. Let us consider $\Gamma = \Pi_{1\leq j\leq n} P^j$ and $\alpha_{tInt}(\Gamma) = \Pi_{1\leq j\leq n}\alpha^j_{tInt}(P^j)$, where n is the cardinality of the senders processes.

Proposition 8 *System* $\Gamma \in tGNDC^{\alpha_{tInt}(\Gamma)}_{\leq_{ttrace}}$.

The result follows by application of Propositions 2 and 7.

We note that, in order to have timed integrity on the messages m_i, μTESLA must ensure timed secrecy on the keys K_i. Indeed, we could also check explicitly timed secrecy on the keys with the same machinery.

We also note that, with respect to the previous analyzed case study (EMSS) here we have some results about not only an arbitrary number of receivers, but also an arbitrary number of senders.

[10]Note that $\mathbf{0}'||\alpha_{tInt}(P^q) \leq_{ttrace} \alpha_{tInt}(P^q)$.

Chapter 4

The Team Automata Chapter

The Team Automata point of view in the formal modelling and analysis of security and privacy

4.1 Introduction

THREAD OF THE CHAPTER. Recent years have seen an increasing interest in the use of automata-based formalisms for the specification and verification of security properties in communication protocols [GOR02, LMST03, Lyn99, Ohe03, OL02]. This chapter is devoted to show the results obtained in using Team Automata — an extension of Input/Output (IOA) automata [LT89]—in the modelling of secure multicast communication [BLP03, BLP04b] and secure mobile agents [EP04] and in the analysis of some properties. In particular, we analysed a functional property called integrity, *i.e.*, robustness against modification of messages, and a privacy property.

TEAM AUTOMATA. Team Automata (TA) have originally been introduced in the context of Computer Supported Cooperative Work (CSCW) [BEKR03, Ell97, Kle03].

They are inspired by—and form an extension of—Input/Output automata (IOA) [LT89]. Like IOA, TA form a flexible framework for modelling communication between components of distributed and reactive systems. They model the logical architecture of a system by describing it solely in terms of an automaton, the role of actions, and synchronizations between these actions. A TA is composed of component automata (CA), which are ordinary automata without final states

and with a distinction of their actions into input, output and internal actions. The only difference between a CA and an IOA is that IOA are by definition *input enabled*: in each state it must be possible to execute every input action. The crux of composing a TA is to define the way in which its constituting CA communicate by synchronizations. Whereas IOA are constructed according to a single and very strict method of composing automata, in effect resulting in composite automata that are uniquely defined by their constituents, there is no such thing as the unique TA composed over a given set of CA. Rather, a whole range of TA, distinguishable only by their synchronizations, can be composed over this set of CA. In particular, contrary to the case of IOA, in TA also output actions may be synchronized upon.

The rigorous setup of these frameworks allows one to formulate and verify general and specific logical properties of complex (distributed, reactive) systems in a mathematically precise way. In realistically large computer systems, security is a big issue, and these frameworks allow formal proofs of correctness of its design. Moreover, such a formal approach forces one to unambiguously describe one's design and it may suggest new approaches not seen otherwise. The particular characteristics of TA with respect to IOA were showed to be useful in specific circumstances, two of which we describe next. In [BEKR01], the synchronization of output actions was used to define so-called *peer-to-peer* and *master-slave* synchronizations. These are two important CSCW phenomena, which were thus introduced with a clear practical motivation in mind. Neither of them can however be distinguished in IOA.

CONTRIBUTIONS I. We use the freedom of choosing the synchronizations of a TA over a set of CA to define a so-called *multicast* composition operator $\|^J$ as a one-to-J synchronization between a sender and a subset J of the total set of receivers. This notion cannot be distinguished in IOA. In particular, we show the potential of TA for modelling secure multicast and broadcast communication. To this aim, TA were used to model an instance of a particular stream signature protocol. The one-to-many and one-to-all communications that are so typical of multicast and broadcast communications, could be captured by TA in a native way as synchronizations between the set of CA constituting a TA [BLP03].

We attempt to provide TA with a framework for security analysis [BLP04b]. First, we define an insecure communication scenario for TA, based on adding a so-called *most general intruder* (a kind of Dolev-Yao model, that is commonly used in security analysis, [DY83]) to a TA model of a secure communication protocol. The intruder is modelled as an active agent able to influence the communication among honest agents. Secondly, we reformulate in terms of TA the Generalized

Non-Deducibility on Compositions (GNDC) schema of [FM99] (recalled in the Preliminaries) and, subsequently, a compositional analysis strategy is described for it, which can be used for verifying security properties in the communication protocol modelled by the scenario. Thirdly, we apply this framework to show that the property of integrity, *i.e.*, a sort of robustness against packet modification, is guaranteed for a case study in which TA model a particular instance of the Efficient Multi-chained Stream Signature (EMSS) protocol family of [PCTS00]. (In a previous chapter of this thesis it has been shown how the same protocol can be modeled by means of process algebras and the same property has been analyzed. Adopting the same case study should help in facilitating an easy comparison for those familiar with other approaches.)

RELATED WORK. The approach of using an automata-based formalism for the specification and verification of properties in the field of security is not unique, but has become very popular in recent years [GOR02, LMST03, Lyn99, Ohe03, OL02, Sch00]. We briefly describe some approaches closest to those proposed in this chapter.

In [Sch00], so-called *Security Automata* are defined as a form of Büchi automata, similar to ordinary (finite) automata, and applied to a simple access control model. Similar to composition of IOA and TA, so-called *conjunction security automata* are defined. It remains to see whether also complex access control policies with delegation and revocation can be modelled by security automata. Such policies have been taken into account in [BEKR01], where the potential of TA for capturing information security and protection structures, and critical coordinations between these structures, is demonstrated. On the basis of a spatial access metaphor, various known access control strategies are formally specified in terms of synchronizations in TA. Moreover, in [tBB] an attempt was initiated to validate some of the resulting specifications with the model checker SPIN [Hol03]. These matters are not however further discussed in this chapter.

In [Lyn99], an experiment involving the combination of simple shared-key communication with the Diffie-Hellman key distribution protocol [DH76] is modelled and proved correct using IOA. As noted by the author herself, a limitation of this approach is the fact that the protocol allows only purely passive eavesdroppers to listen in on the communication. This choice simplifies the formulation of compositional results, as an eavesdropper cannot change the course of communication, *e.g.* by conducting a communication in which it pretends to be an honest participant. The approach does provide attractive compositional reasoning techniques. In this chapter we discuss how the TA approach can be exploited in order

to overcome the limitations of [Lyn99] (see also [BLP04b]). In particular, the proposed insecure scenario shows an active intruder is *a priori* able to listen to public communication between honest components, but also to inject fake messages back to the network.

Finally, another related approach can be found in [Ohe03, OL02], where Interacting State Machines (ISMs)—yet another extension of IOA—are introduced and applied to security analysis. In fact, ISMs are used to model and analyze the classic Needham-Schroeder public-key authentication protocol in the version fixed by Lowe [Low96]. A strong point of this approach is the fact that it allows simultaneous input/output and machine assistance. ISMs are defined, and theorems proved, in the theorem prover Isabelle/HOL [NPW02]. What is missing are solid techniques for compositional reasoning over more complex communication protocols. We define and prove, for a particular insecure communication scenario, a compositional analysis strategy for a TA setting. That strategy allows, to some extent and under some conditions, to verify properties of a formal specification by separately analysing its components ([BLP04b]).

PRIVACY WITHIN SECURE AGENTS. Along with (almost) general analysis techniques for TA, the chapter will present some results in the area of secure agents and privacy properties. Here, a brief introduction to the overall scenario. Agent technology is assuming a central role in various areas of computer science. Mobile agents are indeed a powerful tool for limiting data traffic or managing remote service provision. But since mobile agents are software meant to run on foreign hosts, various security issues arise in their respect. On the one side, hosts must be protected from non trusted agents that might carry malicious code. This is the easy side of the question, and is addressed with computer security techniques. In contrast, it is much harder to protect a mobile agent from a hostile environment. While an agent executes on a host, its code must be in the clear; if it needs to use sensitive data that it carries along, this data must be in the clear as well. If it is stored in an encrypted form, it must be decrypted prior to use, and therefore the appropriate decryption key must be available to the agent. This suggests that usual cryptographic tools cannot protect an agent from being robbed or spied upon.

But, although some vulnerabilities cannot be eliminated (the agent can be killed or bogus data can be supplied to it), it would be very appealing to provide data privacy and integrity mechanisms to mobile agents. Enhancing them with security features can result in a very powerful and effective way of handling services on potentially hostile resources.

Because of the wide range of applications that can be imagined for a secure

agent, the issue is currently a hot topic in research. Recent approaches look for a solution by carrying the idea of encryption to an unusual level; briefly stated, although the agent code itself remains in the clear, the function computed by the agent is transformed so that the agent's behaviour is incomprehensible to an observer that doesn't have a key for interpretation. The first proposal in this direction is due to Sander and Tschudin [ST98]. Their technique has been generalized by Cachin et al. [CCKM00] who use the idea of garbled circuits by Yao [Yao86]. It must be noted that these approaches are still pioneer solutions to the security problem posed above, in that the security goals achieved are still very restrictive and the agent model to which they apply is somewhat awkward.

CONTRIBUTIONS II. As a first step towards a comprehension of the potentialities of these methods, and of intrinsic limitations of software agents from the point of view of security, Team Automata are used to formalize the protocol of Cachin et al. [CCKM00]. The latter is based on the idea of entrusting data to an agent in the form of a circuit that evaluates to a single output. The circuit is obtained as a cascade of components each one constructed by one of the hosts visited by the agent. In [CCKM00], it is proven that the protocol preserves the privacy of all actors' inputs.

Part of this chapter is devoted to showing the recent contributions within a TA based analysis of this privacy property. To the best of our knowledge, TA have not been used before in the study of privacy.

Our analysis also has the merit of providing a high level model of the actors' behaviour and interaction, abstracting out from cryptographic details, giving a clearer insight of the protocol. Our model, interestingly, naturally represents the agent as a set of actions as opposed to an entity *per se*. This suggests that the protocol does not respect the object oriented spirit of agents. On the other hand, taking the perspective of the agent's source, it models the source's view of the system according to the intuition that the source delegates tasks fully to the agent. Moreover, our effort contributes to testify the expressive power and modelling capabilities of team automata.

SUMMARY. This chapter is organized as follows. First, we formally define the framework for TA with multicast-broadcast communication (Section 4.2). Section 4.3 presents the model of the deterministic (1,2) EMSS by TA (this security multicast protocol has been already presented and modeled through process algebras in one of the previous chapters). In Section 4.4 we describe an insecure communication scenario for team automata. In Section 4.5 we reformulate the GNDC schema in terms of team automata and enrich the insecure scenario with

a compositional analysis strategy. We subsequently apply this in Section 4.6 by verifying integrity of the instance of EMSS modeled in Section 4.2. In section 4.7 we give a high level description of the protocol by Cachin *et al.* and recall the relevant results about garbled circuits and the protocol itself. Section 4.8 is devoted to show the model of the protocol and the formal analysis of its privacy features.

4.2 Multicast/broadcast communication in TA

In [BEKR03] several fixed strategies for choosing the synchronizations of a TA were defined, each leading to a uniquely defined TA. These strategies fix the synchronizations of a TA by defining, per action a, certain conditions on the a-transitions to be chosen from $\Delta_a(\mathcal{S})$, thus determining a unique subset of $\Delta_a(\mathcal{S})$ as the set of a-transitions of the TA. Such subsets are referred to as *predicates* for a. Once predicates have been chosen for all actions, the TA over $\mathcal{S}$ defined by these predicates is unique. In [BLP03], new predicates specifically for modelling multicast/broadcast communication in TA have been introduced. Here, we recall them.

Definition 20 *Let $a \in \bigcup_{i\in\mathcal{I}} \Sigma_i$ and let $J \subseteq \mathcal{I}$. The predicate J-*cast *for a in $\mathcal{S}$, denoted by $\mathcal{R}_a^J(\mathcal{S})$, is defined as $\mathcal{R}_a^J(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid [\forall j \in J : [a \in \Sigma_j \wedge a en_{\mathcal{C}_j} proj_j(q))] \Rightarrow proj_j{}^{[2]}(q,q') \in \delta_{j,a}] \wedge [\forall i \in \mathcal{I}\setminus J : [proj_i(q) = proj_i(q')]\}$.*

The predicate J-cast thus contains *all and only* those a-transitions from $\Delta_a(\mathcal{S})$ in which every CA from J participates *whenever* a is currently enabled in that CA, while none of the other CA does. These predicates thus model multicast communication between CA. Obviously, the $\mathcal{I}$-cast predicate models broadcast communication between CA. Hence, we may also refer to it as the *broadcast* predicate.[1]

Definition 21 *Let $J \subseteq \mathcal{I}$, let $\mathcal{R}_a^J(\mathcal{S}) \subseteq \Delta_a(\mathcal{S})$ for all $a \in \bigcup_{i\in\mathcal{I}} \Sigma_i$, and let $\mathcal{R}^J = \{\mathcal{R}_a^J(\mathcal{S}) \mid a \in \bigcup_{i\in\mathcal{I}} \Sigma_i\}$. Then $\mathcal{T} = (Q, (\Sigma_{out}, \Sigma_{inp}, \Sigma_{int}), \delta, I)$ is* the $\mathcal{R}^J$-TA *over $\mathcal{S}$, denoted by $||^J\mathcal{S}$, if $\delta_a = \mathcal{R}_a^J(\mathcal{S})$, for all $a \in \bigcup_{i\in\mathcal{I}} \Sigma_i$.*

Each $\mathcal{R}^J$-TA over $\mathcal{S}$, with $J \subseteq \mathcal{I}$, is also called *the J-cast TA* over $\mathcal{S}$ because it is the unique TA with the following property: the addition of any synchronization results in a TA that contains a synchronization in which a CA from J in which a is currently enabled does not participate. Furthermore, the $\mathcal{R}^{\mathcal{I}}$-TA over $\mathcal{S}$ is also called *the broadcast TA* over $\mathcal{S}$ and it may also be denoted by $|||\mathcal{S}$.

[1] The broadcast predicate is called *is-state-indispensable* in [BEKR03].

4.3 A case study: the EMSS protocol modelled by TA

Among the multicast protocols presented in Chapter 3, here we remind EMSS, exploiting a combination of hash functions and digital signatures for achieving authentication of streams and some robustness against packet loss.

In this section, we show how to specify the deterministic (1,2) schema of the EMSS protocol by team automata. Again, we specify team automata in the way that I/O automata are commonly defined [Lyn99, LT89], *i.e.*, by means of states, actions and transitions. As already done for the specification of Top_C^{ϕ}, we omit the precondition (effect) of an action when it is true.

The sender S of the stream is modeled by a CA $\mathcal{T}_S$ and the set $\{R_n \mid n \geq 1\}$ of receivers by n copies of a CA $\mathcal{T}_R$. $\mathcal{T}_S$ uses its private key $sk(\mathcal{T}_S)$ and a public key $pk(\mathcal{T}_S)$ to perform regular digital signature operations. Let Messages denote the set $\{m_0, m_1, \ldots, m_{last}\}$ of meaningful payloads. Then $\mathcal{T}_S$ uses the hash function h : Messages $\rightarrow$ Hashed, while $\mathcal{T}_R$ uses the hash function $\bar{h} = h$. Moreover, $\mathcal{T}_S$ uses the function $s : 2^{\mathsf{Hashed}_{\mathcal{T}_S}} \rightarrow \mathsf{Signed}_{\mathcal{T}_S}$, defined by $s(H) = H_{sk(\mathcal{T}_S)}$, to sign sets of hashed messages with its private key $sk(\mathcal{T}_S)$, whereas $\mathcal{T}_R$ uses the function $\bar{s} : \mathsf{Signed}_{\mathcal{T}_S} \rightarrow \{\mathsf{true}, \mathsf{false}\}$ and the public key $pk(\mathcal{T}_S)$ to verify whether or not a set of hashed messages was signed by $\mathcal{T}_S$.

$\mathcal{T}_S$

Actions

Inp: $\varnothing$

Out: $\{\overbrace{\langle m_0, \varnothing, \varnothing\rangle}^{P_0}, \overbrace{\langle m_1, h(P_0), \varnothing\rangle}^{P_1}\} \cup \{\overbrace{\langle m_i, h(P_{i-1}), h(P_{i-2})\rangle}^{P_i} \mid 2 \leq i \leq \mathit{last}\} \cup \{\underbrace{\langle \{h(P_{\mathit{last}}), h(P_{\mathit{last}-1})\}_{sk(\mathcal{T}_S)}\rangle}_{P_{\mathit{sign}}}\}$

Int: $\{\mathit{Hash}_i \mid 0 \leq i \leq \mathit{last}\} \cup \{\mathit{Sign}\}$

States

sent $\subseteq$ Messages, hashed $\subseteq$ Hashed, signed $\subseteq$ Signed, all initially $\varnothing$

Transitions

P_0

Eff: sent := sent $\cup \{P_0\}$

Hash$_i$, $0 \leq i \leq$ *last*
Pre: $P_i \in \mathsf{sent} \wedge h(P_i) \notin \mathsf{hashed}$
Eff: $\mathsf{hashed} := \mathsf{hashed} \cup \{h(P_i)\}$

P_1
Pre: $h(P_0) \in \mathsf{hashed} \wedge P_1 \notin \mathsf{sent}$
Eff: $\mathsf{sent} := \mathsf{sent} \cup \{P_1\}$

P_i, $2 \leq i \leq$ *last*
Pre: $\{h(P_{i-1}), h(P_{i-2})\} \subseteq \mathsf{hashed} \wedge P_i \notin \mathsf{sent}$
Eff: $\mathsf{sent} := \mathsf{sent} \cup \{P_i\}$

Sign
Pre: $h(P_{last}) \in \mathsf{hashed} \wedge s(\{h(P_{last}), h(P_{last-1})\}) \notin \mathsf{signed}$
Eff: $\mathsf{signed} := \mathsf{signed} \cup \{s(\{h(P_{last}), h(P_{last-1})\})\}$

P_{sign}
Pre:$\{h(P_{last}), h(P_{last-1})\}_{sk(\mathcal{T}_S)} \in \mathsf{signed} \wedge P_{sign} \notin \mathsf{sent}$
Eff: $\mathsf{sent} := \mathsf{sent} \cup \{P_{sign}\}$

Clearly $\mathcal{T}_S$ has no input behaviour, while its output behaviour $\mathbf{B}_{\mathcal{T}_S}^{\Sigma_{out}}$ consists of all prefixes of $P_0P_1 \cdots P_{last}P_{sign}$. To send the packets $P_0, P_1, \ldots, P_{last}, P_{sign}$ in this order, $\mathcal{T}_S$ must perform some internal computations. This is reflected by its internal behaviour $\mathbf{B}_{\mathcal{T}_S}^{\Sigma_{int}}$ consisting of all prefixes of $Hash_0Hash_1 \cdots Hash_{last}Sign$.

We continue with the specification of $\mathcal{T}_R$. It is capable of receiving as input behaviour all packets $P_0, P_1, \ldots, P_{last}, P_{sign}$, built over the set $\mathsf{Payloads}'$ of variables m_i' that should contain the meaningful payloads m_i. Upon receiving P_i, $\mathcal{T}_R$ verifies whether it has received P_{i-1}. First consider that $\mathcal{T}_R$ indeed received P_{i-1}. Then it extracts the hash $h(P_{i-1})$ from P_i, computes the hash $\bar{h}(P_{i-1})$, and compares these two hashes. If they are equal, then the variable m_{i-1}' that should contain the verifiable payload m_{i-1} is extracted from P_{i-1}. Otherwise $\mathcal{T}_R$ has no output behaviour.

Secondly, consider that $\mathcal{T}_R$ did not receive P_{i-1}. Then it verifies whether it received P_{i-2}. If it did not, then $\mathcal{T}_R$ concludes that it is unable to check the hashes of either P_{i-1} or P_{i-2}, so it goes on to verify whether it did receive P_{i+1}. If $\mathcal{T}_R$ did receive P_{i-2}, then it extracts the hash $h(P_{i-2})$ from P_i, computes the hash $\bar{h}(P_{i-2})$,

and compares the two hashes. If they are equal, then the variable m'_{i-2} that should contain the verifiable payload m_{i-2} is extracted from P_{i-2}. Otherwise $\mathcal{T}_R$ has no output behaviour.

Eventually $\mathcal{T}_R$ receives the signature packet P_{sign} (we assume that P_{sign} is always received, but in the specification of $\mathcal{T}_R$ we sometimes check if P_{sign} has already been received to avoid a transition to take place *before* P_{sign} has actually been received), after which it verifies the accompanying digital signature (we assume that $\mathcal{T}_R$ has previously retrieved the public key $pk(\mathcal{T}_S)$ corresponding to the private key $sk(\mathcal{T}_S)$), before repeating the above procedure. The verification of the signature allows $\mathcal{T}_R$ to have guarantees on the integrity of the stream of verifiable payloads collected in xtractedM, which is consequently sent to the application level as the output behaviour of $\mathcal{T}_R$.

Note that in the specification of $\mathcal{T}_S$ we explicitly modelled that each of its actions is enabled only once during a computation, thus prohibiting loops. For example, as soon as $\mathcal{T}_S$ has sent P_0, then this action's precondition $P_0 \notin \mathsf{sent}$ prohibits this action to be executed again. For the sake of readability, we omit the addition of such preconditions to the specification of $\mathcal{T}_R$, but implicitly assume that its actions are executed only once during a computation.

$\mathcal{T}_R$

Actions

Inp: $\{\overbrace{\langle m'_0, \varnothing, \varnothing\rangle}^{P_0}, \overbrace{\langle m'_1, h(P_0), \varnothing\rangle}^{P_1}\} \cup \{\overbrace{\langle m'_i, h(P_{i-1}), h(P_{i-2})\rangle}^{P_i} \mid 2 \leq i \leq \mathit{last}\} \cup \{\underbrace{\langle\{h(P_{\mathit{last}}), h(P_{\mathit{last}-1})\}_{sk(\mathcal{T}_S)}\rangle}_{P_{sign}}\}$

Out: $\mathsf{Payloads}'$

Int: $\{\mathit{XtractH}_i, \mathit{XtractM}_i, \mathit{Hash}_i \mid 0 \leq i \leq \mathit{last}\} \cup \{\mathit{Verify}, \mathit{Stream}\}$

States

$\mathsf{received}, \mathsf{xtractedM} \subseteq \mathsf{Payloads}'$, $\mathsf{xtractedH}, \mathsf{hashed} \subseteq \mathsf{Hashed}$, all initially $\varnothing$

$\mathsf{verified}, \mathsf{send} \subseteq \{\mathsf{true}, \mathsf{false}\}$, both initially false

Transitions

$P_i,\ 0 \leq i \leq \mathit{last}$

Eff: $\mathsf{received} := \mathsf{received} \cup \{P_i\}$

$\mathit{XtractH}_{i,1},\ 1 \leq i \leq \mathit{last}$

Pre: $\{P_{i-1}, P_i\} \subseteq$ received
Eff: xtractedH $:=$ xtractedH $\cup \{h(P_{i-1})\}$

XtractH$_{i,2},\ 2 \leq i \leq$ *last*
Pre: $[\{P_{i-2}, P_i\} \subseteq$ received$] \wedge [P_{i-1} \notin$ received$]$
Eff: xtractedH $:=$ xtractedH $\cup \{h(P_{i-2})\}$

P_{sign}
Eff: received $:=$ received $\cup \{P_{sign}\}$

Verify
Pre: $[P_{sign} \in$ received$] \wedge [\bar{s}(\{h(P_{last}), h(P_{last-1})\}_{sk(\mathcal{T}_S)}) =$ true$]$
Eff: verified $:=$ true

XtractH$_{sign,1}$
Pre: $[\{P_{last}, P_{sign}\} \subseteq$ received$] \wedge [$verified $=$ true$]$
Eff: xtractedH $:=$ xtractedH $\cup \{h(P_{last})\}$

XtractH$_{sign,2}$
Pre: $[\{P_{last-1}, P_{sign}\} \subseteq$ received$] \wedge [P_{last} \notin$ received$] \wedge [$verified $=$ true$]$
Eff: xtractedH $:=$ xtractedH $\cup \{h(P_{last-1})\}$

Stream
Pre: $[[m'_{last} \in$ xtractedM$] \vee [[m'_{last-1} \in$ xtractedM$] \wedge [P_{last} \notin$ received$]]]$
$\wedge[$verified $=$ true$]$
Eff: send $:=$ true

XtractM$_i,\ 0 \leq i \leq$ *last*
Pre: $[h(P_i) \in$ xtractedH$] \wedge [\bar{h}(P_i) \in$ hashed$] \wedge [\bar{h}(P_i) = h(P_i)]$
Eff: xtractedM $:=$ xtractedM $\cup \{m'_i\}$

Hash$_i,\ 0 \leq i \leq$ *last*
Pre: $h(P_i) \in$ xtractedH
Eff: hashed $:=$ hashed $\cup \{\bar{h}(P_i)\}$

m'_0
Pre: [send $=$ true$] \wedge [m'_0 \in$ xtractedM$]$
Eff:xtractedM $:=$ xtractedM $- \{m'_0\}$

$m'_i,\ 1 \leq i \leq last$

Pre: $[\mathsf{send} = \mathsf{true}] \wedge [m'_i \in \mathsf{xtractedM}] \wedge [\{m'_k \mid 0 \leq k < i\} \cap \mathsf{xtractedM} = \varnothing]$

Eff: $\mathsf{xtractedM} := \mathsf{xtractedM} - \{m'_i\}$

Clearly the input behaviour $\mathbf{B}_{\mathcal{T}_R}^{\Sigma_{inp}}$ of $\mathcal{T}_R$ consists of all prefixes of all possible permutations of $P_0 P_1 \cdots P_{last} P_{sign}$. When $\mathcal{T}_R$ subsequently actually receives the packets $P_0, P_1, \ldots, P_{last}, P_{sign}$ in this particular order, then $\mathcal{T}_R$ is able to perform a series of internal computations, which is reflected by the fact that its internal behaviour $\mathbf{B}_{\mathcal{T}_R}^{\Sigma_{int}}$ contains $XtractH_{1,1} Hash_0 XtractM_0 XtractH_{2,1} Hash_1 XtractM_1 \cdots XtractH_{last,1} Hash_{last-1} XtractM_{last-1} VerifyXtractH_{sign,1} Hash_{last} XtractM_{last} Stream$ as well as other traces representing other orders of performing these internal computations. Finally, the output behaviour $\mathbf{B}_{\mathcal{T}_R}^{\Sigma_{out}}$ of $\mathcal{T}_R$ consists of all prefixes of $m'_0 m'_1 \cdots m'_{last}$.

Now the max-ai team automaton over $\{\mathcal{T}_S, \mathcal{T}_R^{(i)} \mid 1 \leq i \leq n\}$, denoted by $\mathcal{T}_{EMSS}$, is defined as

$$\mathcal{T}_{EMSS} = |||\,\{\mathcal{T}_S, \mathcal{T}_R^{(i)} \mid 1 \leq i \leq n\},$$

which formalizes the EMSS protocol. Note that $\mathcal{T}_{EMSS}$ has no input actions, while it has the union of the output (internal) actions of $\mathcal{T}_S$ and the $\mathcal{T}_R$'s as its output (internal) actions.

In the previous section, the reader has been reminded about the definitions of the broadcast and multicast operators given in [BLP03]. To apply those operators to the EMSS case study leads to the following.

Let $\mathcal{C}_1 = \mathcal{T}_S$ and let $C_i = \mathcal{T}_R$,[2] for all $2 \leq i \leq n+1$. Then the $\{1\}$-cast TA over $\{\mathcal{C}_i \mid 1 \leq i \leq n+1\}$ is essentially the same as $\mathcal{T}_S$. Since it has the same output behaviour as $\mathcal{T}_S$, it thus models multicast/broadcast communication with full packet loss. The K-cast TA over $\{\mathcal{C}_i \mid 1 \leq i \leq n+1\}$, where $K \supset \{1\}$, has the union of the output (internal) actions of $\mathcal{T}_S$ and $\mathcal{T}_R$ as its output (internal) actions and it has no input actions. The fact that its output behaviour consists of all prefixes of $P_0 P_1 \cdots P_{last} P_{sign} m_0 m_1 \cdots m_{last}$ implies that it models multicast/broadcast communication.

Finally, we illustrate how we can model multicast/broadcast communication with some packet loss by varying the type of synchronization per output action.

[2] Strictly spoken, the internal actions of each $\mathcal{T}_R$ must be indexed to satisfy the composability condition.

Assume that $\mathcal{T}_S$ performs a multicast communication with two receivers, viz. the jth and the kth receiver, and let $L = \{1\} \cup \{j, k \mid 2 \leq j < k \leq n+1\}$. If there would be no packet loss, then we would compose the L-cast TA over $\{\mathcal{C}_i \mid 1 \leq i \leq n+1\}$. Let Σ be the alphabet of this TA. Then we recall that the L-cast TA may also be called the $\mathcal{R}^L$-TA or—even more detailed—the $\{\mathcal{R}^L_a \mid \forall\, a \in \Sigma\}$-TA over $\{\mathcal{C}_i \mid 1 \leq i \leq n+1\}$. Next we assume that there is some packet loss, viz. the jth receiver does not receive P_1 and the kth receiver does not receive P_{last-1}. To reflect this packet loss, we would compose the $(\{\mathcal{R}^L_a \mid \forall\, a \in (\Sigma - \{P_1, P_{last-1}\})\} \cup \{\mathcal{R}^{L-\{j\}}_{P_1}\} \cup \{\mathcal{R}^{L-\{k\}}_{P_{last-1}}\})$-TA over $\{\mathcal{C}_i \mid 1 \leq i \leq n+1\}$.

4.4 An insecure communication scenario for TA

In this section we adopt a generic communication protocol in order to obtain an insecure communication scenario for team automata in which to analyze security properties.

We assume all actions to be built over a first order signature σ, where predicate symbols are seen as communication channels and atomic formulae as messages. We assume the function symbols in σ to contain at least the ones that we will use in the sequel, that are: the symbols denoting encryption and pairing, *e.g.*, $\{\_\}\_$ and $\langle \_, \_ \rangle$; those denoting hashing, *e.g.*, $h(\_)$; and those indicating the secret and public key, *e.g.*, $sk(\_)$ and $pk(\_)$. We let m, m' range over the set Messages of atomic formulae and c, c' over the set Channels of predicate symbols. In the sequel `Eve`, `Eve`$'$, `Pub`, `Pub`$'$, `Reveal`, and `Reveal`$'$ will be used as particular predicate names. Every action will thus be written as $c(m)$, denoting message m sent over channel c. Given a set $M \subseteq$ Messages of messages, we define $c(M) = \{c(m) \mid m \in M\}$. Given a set C of predicate names we define $C(M) = \{c(m) \mid m \in M,\ c \in C\}$. Finally, with a little abuse of notation, we will also write C as a shortcut for the set C(Messages).

We abstract from the cryptographic details concerning the operations according to which messages can be encrypted, decrypted, hashed, paired, *et cetera*, but we assume the presence of an inference system (defined by a derivation operator $\vdash$) that implements these operations. By applying (cryptographic) operations from this inference system to a set M of messages, a new set $\mathcal{D}(M) = \{m \mid M \vdash m\}$ of messages (usually called the *deduction set*) can be obtained. This approach is standard in the analysis of (cryptographic) communication protocols [CJM00, FM99, LGL03, Lyn99].

In the sequel we assume a cryptographic communication protocol specifica-

tion involving two roles, *viz.* an *initiator* $\mathcal{T}_S$ and a *responder* $\mathcal{T}_R$. Rather than a direct communication between $\mathcal{T}_S$ and $\mathcal{T}_R$, we assume all the communication to flow through an *insecure channel* (cf. Figure 4.1). This insecure channel may release some messages to an *intruder* which, in its turn, can either listen to or modify (fake) the messages passing through this channel. The insecure channel automaton should be seen as a completely passive pipe. Furthermore, it has been introduced for modeling reasons, and it acts as the door for the intruder automaton to interact with the other participants. This automaton has been introduced mainly for *structural* reasons, given the absence, in the TA world, of any structural notion of channels names over which the automata can synchronize. Automata just synchronize on actions. To give a short, and naive, comparison with process algebras, in that world a third party may synchronize with others by synchronizing on the same channel (performing a complementary action, *i.e.*, a send or a receive action). Instead, in the presented architecture, the way for the intruder to listen to and inject new messages back is through this added automaton.

When verifying security properties for (cryptographic) communication protocols, it is indeed quite common to include an additional intruder (*à la* Dolev-Yao [DY83]) component that is supposed to be malicious and whose aim is to subvert the protocol's correct behaviour. A protocol specification is consequently considered secure with respect to a security property if it satisfies this property despite the presence of the intruder. Based on the approach of [Lyn99], the insecure channel and the intruder are modelled by team automata $\mathcal{T}_{IC}$ and $\mathcal{T}_X$. We thus propose a framework of four types of team automata:

1. $\mathcal{T}_S$ plays the role of the protocol's initiator,
2. $\mathcal{T}_R$ plays the role of the protocol's responder,
3. $\mathcal{T}_{IC}$ plays the role of the insecure channel, and
4. $\mathcal{T}_X$ plays the role of the active and malicious intruder.

We do not explicitly model the team automata of our framework, but we informally describe them by their interactions. More precisely, we let the initiator and the responder communicate with the insecure channel through disjoint sets of actions Σ^S_{com} and Σ^R_{com}, respectively, such that a direct communication between them is impossible. The $\mathcal{T}_{IC}$, in its turn, can interact with the intruder only through a distinct set Σ^I_{com} of actions. Finally, some particular actions may be used by an honest role in order to reveal some information to the outside concerning, *e.g.*,

a state reached during a run of the protocol. In Figure 4.1 we have depicted the insecure communication scenario for team automata sketched above.

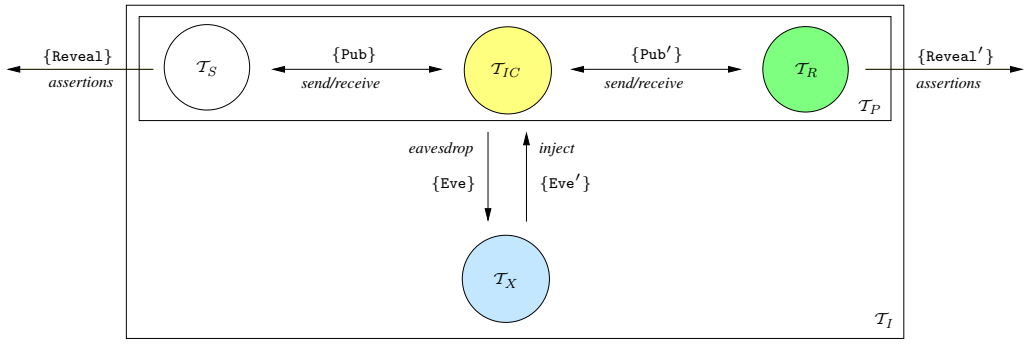

Figure 4.1: An insecure communication scenario for team automata.

We let $\mathcal{T}_P$ denote the team automata representing our protocol specification in the absence of the intruder. We thus define $\mathcal{T}_P$ to be the max-ai team automaton over $\{\mathcal{T}_S, \mathcal{T}_R, \mathcal{T}_{IC}\}$ that is obtained after hiding the actions $\Sigma^P_{com} = \Sigma^S_{com} \cup \Sigma^R_{com}$, *i.e.*, all messages passing through the insecure channel (*e.g.*, $\Sigma^P_{com} = \{\mathtt{Pub}, \mathtt{Pub}'\}$ in Figure 4.1). Hence

$$\mathcal{T}_P = \mathrm{hide}_{\Sigma^P_{com}}(\,|||\,\{\mathcal{T}_S, \mathcal{T}_R, \mathcal{T}_{IC}\}).$$

Recall that, with previously defined notations, the shortcut $\{\mathtt{Pub}, \mathtt{Pub}'\}$ stands for $\{\mathtt{Pub}(m), \mathtt{Pub}'(m) \mid m \in \mathsf{Messages}\}$. By hiding Σ^P_{com}, these actions are no longer available for synchronizations in further team automata composed over $\mathcal{T}_P$. To its environment, $\mathcal{T}_P$ thus appears as a black box, possibly with some output actions Σ^S_{sig} and Σ^R_{sig}—signalling the successful reception of messages. Usually such signals are used only for verification purposes and for the sequel we assume that $\Sigma^S_{sig} \cap \Sigma^R_{sig} = \varnothing$ (*e.g.*, $\Sigma^S_{sig} = \{\mathtt{Reveal}\}$ and $\Sigma^R_{sig} = \{\mathtt{Reveal}'\}$ in Figure 4.1).

We let $\mathcal{T}_I$ denote the team automaton representing our protocol specification in the presence of the intruder. Actions Σ^I_{com} serve as the back-door for intrusion and are added to $\mathcal{T}_{IC}$ (*e.g.*, $\Sigma^I_{com} = \{\mathtt{Eve}, \mathtt{Eve}'\}$ in Figure 4.1). This is exactly what we need to guarantee that the intruder $\mathcal{T}_X$ may communicate with $\mathcal{T}_P$ only through the insecure channel. We thus define $\mathcal{T}_I$ to be the max-ai team automaton over $\{\mathcal{T}_P, \mathcal{T}_X\}$ that is obtained after hiding the actions Σ^I_{com}, *i.e.*, all messages that the intruder can eavesdrop from and inject back into the insecure channel. We thus enforce maximal synchronization between the intruder and the protocol. Hence

$$\mathcal{T}_I = \mathrm{hide}_{\Sigma^I_{com}}(\,|||\,\{\mathcal{T}_P, \mathcal{T}_X\})$$

We have now defined an insecure communication scenario for team automata by composing a secure communication scenario with an intruder.

4.5 Reformulating GNDC in terms of TA

In this section, a re-formulation of GNDC in terms of TA is given ([BLP03, BLP04b]). GNDC, the general schema for defining security properties, has been already presented in its original version (*i.e.*, in terms of process algebras) in Chapter 3.

We begin by instantiating $\mathcal{T}_P$ to be a team automaton modelling communication between an initiator and a set of responders through the use of an insecure channel, in the style of the team automaton $\mathcal{T}_P$ considered in the insecure communication scenario of Section 4.4. To this aim, we let $\mathcal{T}_P$ be specified as $\mathcal{T}_P = \{Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I\}$. Because the original definition of GNDC (Chapter 3, Section 2.4, Definition 13) requires P to communicate with X through the channels contained in C, we require a set $C = C_{inp} \cup C_{out}$ of actions C_{inp} that are input to X and actions C_{out} that are output to X, and for which $C \cap \Sigma^P_{ext} \neq \varnothing$ and $C \cap \Sigma^P_{com} = \varnothing$. This resembles requiring $\mathcal{T}_P$ to be able to communicate with the intruder $\mathcal{T}_X$ only by executing actions in Σ^I_{com} (*e.g.*, $\{\mathtt{Eve}, \mathtt{Eve}'\}$ in Figure 4.1). In the sequel we thus assume C to coincide exactly with Σ^I_{com} and, in particular, C_{inp} with the actions in Σ^I_{com} that are input to $\mathcal{T}_X$ (*e.g.*, $\{\mathtt{Eve}\}$ in Figure 4.1) and C_{out} with the actions in Σ^I_{com} that are output to $\mathcal{T}_X$ (*e.g.*, $\{\mathtt{Eve}'\}$ in Figure 4.1). We are now able to formalize the hostile environment $\mathcal{E}_C$ in terms of team automata as

$$\mathcal{E}_C = \{(Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I) \mid \Sigma_{inp} \subseteq C_{inp},\ \Sigma_{out} \subseteq C_{out}\}. \tag{4.1}$$

In addition, the original GNDC definition requires the initial knowledge of the environment to be bound to a specified set of messages ϕ. This informally means that the environment should be able to produce, by means of only its internal functioning, at most the messages contained in $\mathcal{D}(\phi)$. In terms of team automata this means that a component automaton in the environment, when considered as a stand-alone component, can only execute output actions belonging to $C(\mathcal{D}(\phi))$. This is formally defined by restricting its behaviour to those sequences consisting of solely output actions since—at a more abstract level—these are the sequences that it can produce without receiving any additional messages from outside, *i.e.*, by exploiting only its own knowledge. Let, for a team automaton $\mathcal{T}$,

$Id^{\Gamma}(\mathbf{B}_{\mathcal{T}}) = \{\gamma \in \mathbf{B}_{\mathcal{T}} \mid \gamma \in \Gamma^*\}$, where Γ is a set of actions. Consequently, the *initial knowledge* of $\mathcal{T}$ is defined as $Id^{\Sigma^{\mathcal{T}}_{out}}(\mathbf{B}_{\mathcal{T}})$. The formal definition of the environment $\mathcal{E}_C^{\phi}$ in terms of team automata now thus becomes

$$\mathcal{E}_C^{\phi} = \{\mathcal{X} \in \mathcal{E}_C \mid Id^{\Sigma^{\mathcal{X}}_{out}}(\mathbf{B}_{\mathcal{X}}) \subseteq (C(\mathcal{D}(\phi)))^*\}. \tag{4.2}$$

Finally, we need a behavioral notion of comparison between team automata which abstracts from their internal and communicating actions. Furthermore, we also require that some predefined set C of actions is prevented from being executed by *a fortiori* excluding all sequences in which they do occur. Therefore, we hide those output actions involved in communications and we define the *observational behaviour* (with respect to actions not in C) of the resulting team automata as those subsequences of the (remaining) external behaviour that consist solely of actions not in C.

Definition 22 *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a team automaton over $\mathcal{S}$, let $\Sigma_{com} \subseteq \Sigma_{ext}$ and let $\mathcal{T}' = hide_{\Sigma_{com}}(\mathcal{T})$.*

Then the observational behaviour *of $\mathcal{T}'$ with respect to actions not in C, denoted by $\mathbf{O}^C_{\mathcal{T}'}$, is defined as*

$$\mathbf{O}^C_{\mathcal{T}'} = Id^{\Sigma^{\mathcal{T}'}_{ext} - C}(pres_{\Sigma^{\mathcal{T}'}_{ext}}(\mathbf{B}_{\mathcal{T}'})).$$

As a result we are able to reformulate the definition of GNDC (given in Chapter 3 within a process algebraic framework) in terms of team automata.

Definition 23 *Let $\alpha(\mathcal{T}_P)$ be the expected (correct) behaviour of $\mathcal{T}_P$. Then*

$$\mathcal{T}_P \in GNDC^{\alpha(\mathcal{T}_P)}_{\subseteq} \text{ iff } \forall\, \mathcal{X} \in \mathcal{E}_C^{\phi} : \mathbf{O}^C_{hide_{C}(|||\{\mathcal{T}_P, \mathcal{X}\})} \subseteq \alpha(\mathcal{T}_P).$$

Informally, Definition 23 says that $\mathcal{T}_P$ (*i.e.*, a cryptographic communication protocol specified in the insecure communication scenario) satisfies $GNDC^{\alpha(\mathcal{T}_P)}_{\subseteq}$ if and only if its observational behaviour, despite communicating with any intruder $\mathcal{X}$ through the actions C, is included in $\alpha(\mathcal{T}_P)$ (*i.e.*, the expected correct behaviour of the communication protocol specified by $\mathcal{T}_P$). A significative instance of α is, *e.g.*, $\alpha_{int}(\mathcal{T}_P) = \mathbf{O}^C_{\mathcal{T}_P}$, which will be make precise and used in Section 4.6 to express integrity. Additionally, Definition 23 requires the intruder to be any team automaton able to interact with $\mathcal{T}_P$ through the actions C and with an initial knowledge bound to $\mathcal{D}(\phi)$.

4.5.1 Security analysis strategies for TA

While allowing a uniform approach to specifying security properties, Definition 23 does not provide effective strategies for the security analysis of (cryptographic) communication protocols. The universal quantification over $\mathcal{E}_C^{\phi}$ causes serious problems when deciding whether $\mathcal{T}_P \in GNDC_{\subseteq}^{\alpha(\mathcal{T}_P)}$. However, the theory developed for GNDC in terms of process algebras inspires similar methodologies for team automata.

The Most General Intruder. As a first analysis strategy we give a static characterization of the intruder, not involving the universal quantification of Definition 23.

One reasonable way to avoid the infinite number of checks that the universal quantification would require is to study whether there is an attacker that is more powerful (with respect to a chosen behavioral relation) than all the others. In this way one can reduce the analysis against any environment to an analysis against only one, albeit very powerful, so-called *most general intruder*. From the theory of GNDC [FGM04] we know that a sufficient condition for the existence of such a most general intruder, is to have a behavioral relation that is a *pre-congruence* with respect to the (parallel) composition and restriction operators. Restated in our framework we say that $\lhd$ is a pre-congruence (with respect to $|||$ and hide_C) if for every automaton $\mathcal{T}$, $\mathcal{X}$ and $\mathcal{X}'$ in $\mathcal{E}_C$, whenever $\mathbf{B}_{\mathcal{X}}^C \lhd \mathbf{B}_{\mathcal{X}'}^C$ then $\mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T},\mathcal{X}\})}} \lhd \mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T},\mathcal{X}'\})}}$. It is not difficult to prove that this is true in our case, *viz.*

Lemma 9 *Let $\mathcal{T} = (Q, (\Sigma_{inp}^{\mathcal{T}}, \Sigma_{out}^{\mathcal{T}}, \Sigma_{int}^{\mathcal{T}}), \delta, I)$ be a team automaton and let $\mathcal{X}, \mathcal{X}' \in \mathcal{E}_C$. Then*

$$\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{\mathcal{X}'}^C \textit{ implies } \mathbf{O}^C_{\textit{hide}_{C(|||\{\mathcal{T},\mathcal{X}\})}} \subseteq \mathbf{O}^C_{\textit{hide}_{C(|||\{\mathcal{T},\mathcal{X}'\})}}.$$

Proof. Let $a_1 \cdots a_n \in \mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T},\mathcal{X}\})}}$ and let $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{\mathcal{X}'}^C$. By (4.1), $\Sigma_{ext}^{\mathcal{X}} \subseteq C$ because $\mathcal{X} \in \mathcal{E}_C$. Then by Definition 22, for all $i \in [n]$, $a_i \in \Sigma_{ext}^{\mathcal{T}} - C$. We now use that by definition also all prefixes of $a_1 \cdots a_n$ are included in $\mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T},\mathcal{X}\})}}$ and show by induction that all prefixes of $a_1 \cdots a_n$ are also included in $\mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T},\mathcal{X}'\})}}$. First consider a_1. By Definition 22, either $a_1 \in \mathbf{B}_{\text{hide}_{C(|||\{\mathcal{T},\mathcal{X}\})}}$ or $b_1 \cdots b_m a_1 \in \mathbf{B}_{\text{hide}_{C(|||\{\mathcal{T},\mathcal{X}\})}}$, for some $m \geq 1$ and where, for all $j \in [m]$, b_j is an internal action of $\text{hide}_C(|||\{\mathcal{T}, \mathcal{X}\})$. In both cases, since $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{\mathcal{X}'}^C$ and $a_i \in \Sigma_{ext}^{\mathcal{T}} - C$, for all $i \in [n]$, it follows by Definition 22

that $a_1 \in \mathbf{O}^C_{\mathsf{hide}_{C(|||\{\mathcal{T},\mathcal{X}'\})}}$. Now assume that $a_1 \cdots a_k \in \mathbf{O}^C_{\mathsf{hide}_{C(|||\{\mathcal{T},\mathcal{X}'\})}}$, with $k < n$, and consider $a_1 \cdots a_{k+1}$. Using similar arguments as above and the induction hypothesis it follows that $a_1 \cdots a_{k+1} \in \mathbf{O}^C_{\mathsf{hide}_{C(|||\{\mathcal{T},\mathcal{X}'\})}}$. □

Since $\mathcal{E}^\phi_C \subseteq \mathcal{E}_C$, this lemma holds for $\mathcal{X}, \mathcal{X}' \in \mathcal{E}^\phi_C$ as well. Based on the approach of [FM99] we now define a component automaton Top^ϕ_C, representing the most general intruder mentioned above, in order to circumvent the universal quantification of Definition 23. Recall that the set C of predicates that the intruder uses to interact with the insecure channel is partitioned into C_{inp} and C_{out}, *i.e.*, the channel names the intruder uses to retrieve messages from and to inject messages back into the insecure channel, respectively (*e.g.*, in Figure 4.1, $C_{inp} = \{\mathsf{Eve}\}$ and $C_{out} = \{\mathsf{Eve}'\}$).

We specify Top^ϕ_C in the way that I/O automata are commonly defined [Lyn99, LT89]. Its states are thus defined by the current values of the variables listed under States, while its transitions are defined, per action a, as preconditions (Pre) and effect (Eff), *i.e.*, (q, a, q') is a transition of Top^ϕ_C if the precondition of a is satisfied by$q$, while q' is the transformation of q defined by the effect of a. We omit the precondition of an action when it is true.

Recall that C (C_{inp}, C_{out}) is a shortcut for $C(\mathsf{Messages})$ ($C_{inp}(\mathsf{Messages})$, $C_{out}(\mathsf{Messages})$) and that $C = C_{inp} \cup C_{out}$.

Top^ϕ_C

Actions

Inp: $C_{inp}(\mathsf{Messages})$ Out: $C_{out}(\mathsf{Messages})$ Int: $\varnothing$

States

$\mathsf{received} \subseteq \mathsf{Messages}$, initially ϕ

Transitions

$c(m) \in C_{inp}(\mathsf{Messages})$
Eff: $\mathsf{received} := \mathsf{received} \cup \{m\}$

$c(m) \in C_{out}(\mathsf{Messages})$
Pre: $m \in \mathcal{D}(\mathsf{received})$

The asymmetry between input and output relies on the fact that the automaton has the capability to possibly apply the rules from its inference system to the messages it receives. In other words, Top^ϕ_C could not act as a passive pipe only.

The general way in which Top^ϕ_C is specified implies that its behaviour includes

that of any automaton from $\mathcal{E}_C^\phi$.

Lemma 10 *For all* $\mathcal{X} \in \mathcal{E}_C^\phi$, $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{Top_C^\phi}^C$.

Proof. Let $\mathcal{X} \in \mathcal{E}_C^\phi$. Then (4.2) implies that $\mathcal{X} \in \mathcal{E}_C$ and thus, by (4.1) and the specification of Top_C^ϕ, $\Sigma_{inp}^{\mathcal{X}} \subseteq C_{inp} = \Sigma_{inp}^{Top_C^\phi}$ and $\Sigma_{out}^{\mathcal{X}} \subseteq C_{out} = \Sigma_{out}^{Top_C^\phi}$. From (4.2) and the specification of Top_C^ϕ it now follows immediately that $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{Top_C^\phi}^C$. □

Lemmata 9 and 10 directly imply the following result.

Theorem 2 *For all* $\mathcal{X} \in \mathcal{E}_C^\phi$, $\mathbf{O}_{hide_C(|||\{\mathcal{T}_P,\mathcal{X}\})}^C \subseteq \mathbf{O}_{hide_C(|||\{\mathcal{T}_P,Top_C^\phi\})}^C$.

Together with Definition 23, this gives us the following result.

Corollary 3 *Let* $\alpha(\mathcal{T}_P)$ *be as in Definition 23. Then*

$$\mathcal{T}_P \in GNDC_{\subseteq}^{\alpha(\mathcal{T}_P)} \text{ iff } \mathbf{O}_{hide_C(|||\{\mathcal{T}_P,Top_C^\phi\})}^C \subseteq \alpha(\mathcal{T}_P).$$

Compositionality results. We now describe some compositionality results for the insecure communication scenario of Section 4.4. To begin with, we let

$$\mathcal{T}_1 = \text{hide}_{\Sigma_{com}^P}(|||\{\mathcal{T}_S, \mathcal{T}_{IC}\}) \text{ and } \mathcal{T}_2 = \text{hide}_{\Sigma_{com}^P}(|||\{\mathcal{T}_R, \mathcal{T}_{IC}\}).$$

We then let $\mathcal{T}_P$ be the team automaton defined at the end of Section 4.4, *i.e.*, with $\Sigma_{com}^I = C$ added to $\mathcal{T}_{IC}$.

Now $\mathcal{T}_P$ thus represents the communication scenario in which an initiator and a responder are connected by an insecure channel, but not yet connected to an intruder. If we add the most general intruder, some general compositional results can be proved. To this aim we let

$$\mathcal{T}_1' = \text{hide}_C(|||\{\mathcal{T}_1, Top_C^\phi\}) \text{ and } \mathcal{T}_2' = \text{hide}_C(|||\{\mathcal{T}_2, Top_C^\phi\}).$$

Lemma 11 *Let* $\{m \mid \{c(m) \in \Sigma_{com}^P\} \subseteq \phi$. *Then*

$$\mathbf{O}_{hide_C(|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\},Top_C^\phi\})}^C = \underline{||}_{\{\Sigma^{\mathcal{T}_1'},\Sigma^{\mathcal{T}_2'}\}} \{\mathbf{O}_{\mathcal{T}_1'}^C, \mathbf{O}_{\mathcal{T}_2'}^C\}.$$

Proof. From the way $\mathcal{T}_1'$ and $\mathcal{T}_2'$ are composed it follows that $\Sigma_{ext}^{\mathcal{T}_1'} = \Sigma_{sig}^{S}$ and $\Sigma_{ext}^{\mathcal{T}_2'} = \Sigma_{sig}^{R}$. Now let $\mathcal{T}'' = \text{hide}_C(\,|||\,\{\,|||\,\{\mathcal{T}_1, \mathcal{T}_2\}, \mathit{Top}_C^{\phi}\})$. Then it remains to prove that $\mathbf{O}_{\mathcal{T}''}^{C} = \underline{||}_{\{\Sigma^{\mathcal{T}_1'}, \Sigma^{\mathcal{T}_2'}\}} \{\mathbf{O}_{\mathcal{T}_1'}^{C}, \mathbf{O}_{\mathcal{T}_2'}^{C}\}$. Since $\Sigma_{sig}^{S} \cap \Sigma_{sig}^{R} = \varnothing$, this however follows directly from the fact that $\{m \mid \{c(m) \in \Sigma_{com}^{P}\} \subseteq \phi$, *i.e.*, adding $\mathcal{T}_R$ ($\mathcal{T}_S$) to $\mathcal{T}_1'$ ($\mathcal{T}_2'$) does not change the signals from Σ_{sig}^{S} (Σ_{sig}^{R}) which $\mathcal{T}_S$ ($\mathcal{T}_R$) can output because all messages that $\mathcal{T}_R$ ($\mathcal{T}_S$) can send to $\mathcal{T}_{IC}$ have already been included in the initial knowledge of Top_C^{ϕ}. □

Before continuing, we observe the following property of full synchronized shuffles.

Remark 2 *Let Δ_i, with $i \in [4]$, be alphabets and let $L_i \subseteq \Delta_i^*$. Then clearly $\underline{||}_{\{\Delta_1, \Delta_3\}} \{L_1, L_3\} \subseteq \underline{||}_{\{\Delta_2, \Delta_4\}} \{L_2, L_4\}$ whenever $L_1 \subseteq L_2$ and $L_3 \subseteq L_4$.*

Theorem 3 *If $\mathcal{T}_1 \in GNDC_{\subseteq}^{\mathbf{O}_{\mathcal{T}_1}^{C}}$ and $\mathcal{T}_2 \in GNDC_{\subseteq}^{\mathbf{O}_{\mathcal{T}_2}^{C}}$, then $|||\{\mathcal{T}_1, \mathcal{T}_2\} \in GNDC_{\subseteq}^{\underline{||}_{\{\Sigma^{\mathcal{T}_1}, \Sigma^{\mathcal{T}_2}\}}\{\mathbf{O}_{\mathcal{T}_1}^{C}, \mathbf{O}_{\mathcal{T}_2}^{C}\}}$*

Proof. Let $\mathcal{T}_1 \in GNDC_{\subseteq}^{\mathbf{O}_{\mathcal{T}_1}^{C}}$ and $\mathcal{T}_2 \in GNDC_{\subseteq}^{\mathbf{O}_{\mathcal{T}_2}^{C}}$. Then $\mathbf{O}_{\mathcal{T}_1'}^{C} \subseteq \mathbf{O}_{\mathcal{T}_1}^{C}$ and $\mathbf{O}_{\mathcal{T}_2'}^{C} \subseteq \mathbf{O}_{\mathcal{T}_2}^{C}$ and thus, by Lemma 11 and Remark 2, $\mathbf{O}_{\text{hide}_C(|||\{|||\{\mathcal{T}_1, \mathcal{T}_2\}, \mathit{Top}_C^{\phi}\})}^{C} = \underline{||}_{\{\Sigma^{\mathcal{T}_1'}, \Sigma^{\mathcal{T}_2'}\}}\{\mathbf{O}_{\mathcal{T}_1'}^{C}, \mathbf{O}_{\mathcal{T}_2'}^{C}\} \subseteq \underline{||}_{\{\Sigma^{\mathcal{T}_1}, \Sigma^{\mathcal{T}_2}\}}\{\mathbf{O}_{\mathcal{T}_1}^{C}, \mathbf{O}_{\mathcal{T}_2}^{C}\}$. □

4.6 Analysis of the EMSS protocol by TA

In this section we use the GNDC schema in terms of team automata together with the insecure communication scenario for team automata in order to show that integrity is guaranteed in the deterministic (1,2) schema of the EMSS protocol. Note that this has already been validated in [MPV03], where a CCS-like process algebra was used instead, and the results of the analysis have been shown in Chapter 3. Our goal here is thus to use this particular case study to show the effectiveness of team automata for security analysis. Moreover, their use is not limited to proving integrity, but also security properties like secrecy and entity authentication can be verified. The important aspect is that these properties can be defined through a preorder relation on team automata.

In the following, we refer to the specification given in Section 4.3. We define integrity as the ability of $\mathcal{T}_R$ to accept a message m_i, for any i, only as the ith message sent by $\mathcal{T}_S$. We moreover assume that $\mathcal{T}_R$ signals the acceptance of a stream

of messages as a legitimate one by issuing it as a list of messages through special actions $\{\texttt{Reveal}'\}$. In this case study, $\mathcal{T}_S$ does not output any such signals. We require the expected (correct) observational behaviour $\alpha_{int}(\mathcal{T}_P)$ of $\mathcal{T}_P$ with respect to integrity as $\mathbf{O}^C_{\mathcal{T}_P}$, *i.e.*, as all prefixes of $\texttt{Reveal}'(m_0)\texttt{Reveal}'(m_1)\cdots\texttt{Reveal}'(m_{last})$. Note that following the notation introduced at the beginning of Section 4.4, actions may henceforth be written as composed terms, *e.g.*, $\texttt{Reveal}'(m_0)$'s outermost part $\texttt{Reveal}'$ is a predicate symbol while its innermost part m_0 is an atomic formula.

Consequently, we equip Top^{ϕ}_{C} with an initial knowledge ϕ consisting of all output actions of $\mathcal{T}_S$ and the public key $pk(\mathcal{T}_S)$, *i.e.*, $\phi = \{P_0, P_1, P_i, P_{sign} \mid 2 \leq i \leq last\} \cup \{pk(\mathcal{T}_S)\}$, where $P_0 = \langle m_0, \varnothing, \varnothing\rangle$, $P_1 = \langle m_1, h(P_0), \varnothing\rangle$, $P_i = \langle m_i, h(P_{i-1}), h(P_{i-2})\rangle$, for all $1 \leq i \leq last$, and $P_{sign} = \langle\{h(P_{last}), h(P_{last-1})\}_{sk(\mathcal{T}_S)}\rangle$. We do so solely for analysis reasons, *viz.* in order to enable Top^{ϕ}_{C} to send the correct messages to $\mathcal{T}_R$ through the insecure channel when analyzing the max-ai team automaton over $\mathcal{T}_2$ and Top^{ϕ}_{C} (cf. the forthcoming proof of Proposition 10). Note that the messages contained in its initial knowledge are exactly those that it is anyway able to receive in the max-ai team automaton over $\mathcal{T}_P$ and Top^{ϕ}_{C} by eavesdropping when $\mathcal{T}_S$ sends them through the insecure channel. As is common in security analysis, we rely on the *perfect encryption assumption*, *i.e.*, Top^{ϕ}_{C} cannot deduce $sk(\mathcal{T}_S)$ from ϕ nor can it forge hash and encryption functions by guessing. We also assume that there are no unintended clashes. Hence the observational behaviour of the max-ai team automaton over $\mathcal{T}_1 = \text{hide}_{\Sigma^P_{com}}(|||\{\mathcal{T}_S, \mathcal{T}_{IC}\})$ and Top^{ϕ}_{C} is empty, *i.e.*,

Proposition 9 $\mathcal{T}_1 \in GNDC^{\varnothing}_{\subseteq}$.

Proof. Directly by Corollary 3 because $\mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T}_1, Top^{\phi}_{C}\})}} = \varnothing$. □

The way in which the receiver verifies the messages it receives implies that the observational behaviour of the max-ai team automaton over $\mathcal{T}_2 = \text{hide}_{\Sigma^P_{com}}(|||\{\mathcal{T}_R, \mathcal{T}_{IC}\})$ and Top^{ϕ}_{C} is thus included in the expected observational behaviour $\alpha_{int}(\mathcal{T}_P)$ of $\mathcal{T}_P$ with respect to integrity, *i.e.*,

Proposition 10 $\mathcal{T}_2 \in GNDC^{\alpha_{int}(\mathcal{T}_P)}_{\subseteq}$.

Proof. The proof distinguishes two cases. If $\mathcal{T}_R$ does not output any action $\texttt{Reveal}'(m_i)$, for all payloads m_i, then its empty output behaviour is trivially included in $\alpha_{int}(\mathcal{T}_P)$. Next assume that $\mathcal{T}_R$ outputs actions $\texttt{Reveal}'(m_i)$, for all payloads m_i. To do so, the definition of the EMSS protocol guarantees that $\mathcal{T}_R$

must have verified that P_{sign} was signed with $sk(\mathcal{T}_S)$. Because Top_C^{ϕ} cannot deduce this private key from its initial knowledge and none of the team automata ever outputs this private key, it follows that Top_C^{ϕ} does not know $sk(\mathcal{T}_S)$. Since digital signatures and hash functions cannot be forget by the intruder, the only possibility for $\mathcal{T}_R$ to output the $\texttt{Reveal}'(m_i)$, for all payloads m_i, is that Top_C^{ϕ} has sent $\mathcal{T}_R$ all the correct packages in the correct order. This shows why, in absence of $\mathcal{T}_S$, we had to equip Top_C^{ϕ} with an initial knowledge consisting of all output actions of $\mathcal{T}_S$. Hence $\mathcal{T}_R$ must output all payloads m_i in the correct order and thus $\mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T}_2,Top_C^{\phi}\})}} \subseteq \alpha_{int}(\mathcal{T}_P)$ because the behaviour of any team automaton is prefix closed. □

Finally, after an observation on the composition of team automata that have no internal actions, we can show that integrity is guaranteed in the instance of the EMSS protocol under scrutiny.

Remark 3 *If $\{\mathcal{T},\mathcal{T}\}$ is a composable system, then clearly $\mathbf{B}_{|||\{\mathcal{T},\mathcal{T}\}} = \mathbf{B}_{\mathcal{T}}$.*

Proposition 11 $\mathcal{T}_P \in GNDC_{\subseteq}^{\alpha_{int}(\mathcal{T}_P)}$.

Proof. By Propositions 9 and 10 and Theorem 3 follows $|||\{\mathcal{T}_1,\mathcal{T}_2\} \in GNDC_{\subseteq}^{||_{\{\Sigma^{\mathcal{T}_1},\Sigma^{\mathcal{T}_2}\}}^{\{\mathbf{O}^C_{\mathcal{T}_1},\mathbf{O}^C_{\mathcal{T}_2}\}}} = GNDC_{\subseteq}^{||_{\{\texttt{Reveal}'\}}^{\{\varnothing,\alpha_{int}(\mathcal{T}_P)\}}} = GNDC_{\subseteq}^{\alpha_{int}(\mathcal{T}_P)}$. Then $\mathbf{O}^C_{\text{hide}_{C(|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\},Top_C^{\phi}\})}} \subseteq \alpha_{int}(\mathcal{T}_P)$ by Corollary 3. Since $\mathcal{T}_{IC}$ has no internal actions, $\{\mathcal{T}_{IC},\mathcal{T}_{IC}\}$ forms a composable system. It then follows from Remarks 1 and 3 that $\mathbf{B}_{|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\},Top_C^{\phi}\}} = \mathbf{B}_{|||\{\mathcal{T}_S,\mathcal{T}_R,\mathcal{T}_{IC},Top_C^{\phi}\}} = \mathbf{B}_{|||\{\mathcal{T}_P,Top_C^{\phi}\}}$ and consequently $\mathbf{O}^C_{\text{hide}_{C(|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\},Top_C^{\phi}\})}} = \mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T}_P,Top_C^{\phi}\})}}$ by Definition 22. Hence $\mathbf{O}^C_{\text{hide}_{C(|||\{\mathcal{T}_P,Top_C^{\phi}\})}} \subseteq \alpha_{int}(\mathcal{T}_P)$ and thus, by Corollary 3, $\mathcal{T}_P \in GNDC_{\subseteq}^{\alpha_{int}(\mathcal{T}_P)}$. □

4.7 Garbled circuits and secure agents

The last part of this chapter is devoted to present the results obtained in the area of privacy and secure agents within a TA scenario. Full details in [EP04].

The soundness and privacy properties of the protocol of [CCKM00] follow from mathematical results based on standard cryptographic assumptions. We abstract out from the details and are interested in the behaviour of the actors (the agent, the agent's source and the hosts that are visited by the agent) at a higher

level. Our description of the protocol focuses on the interactions between the actors. We give as much insight on the cryptographic bases on which the protocol relies, as is sufficient for our arguments. We refer to [CCKM00] for a detailed description.

We confine ourselves to the simpler case of the "honest but curious model," in which actors are supposed to follow the protocol correctly, but they might try to learn the private inputs of the other parties.

The goal of the design is a secure agent that travels through many hosts collecting sensitive information and then back to its source; back home, it will be able to deliver the result of a computation on inputs collected at the various hosts together with the source's input. The security feature that the protocol aims at is privacy of all the inputs, that is no party learns the inputs of any other party.

The idea is to combine in a cascade Yao style garbled circuits [Yao86]. The software agent travels from host to host collecting private information in the form of a (portion of) garbled circuit. The circuit (potentially software) is actually data, since it can only be evaluated to a single value once it has been brought back to the agent's source.

4.7.1 Garbled circuits

A garbled circuit is a generalization of a circuit, with the following properties:

- each wire can carry one of two specified random strings (not just bits 0/1), the random strings changing from wire to wire (the pair of strings on each wire have semantical interpretation 0 and 1);

- for each gate a specific computational rule is given, that defines how the random strings in input are to be combined to produce the output, which is again a random string (the semantical interpretation of a gate is a NAND or a XOR, for instance).

The *garbled inputs* are the random strings of each input wire whose semantical value is the value of the corresponding input bit. The *decoding* of the output is a translation of the random strings on the output wires to their semantical meanings. The *garbled circuit* is a description of the structure of the circuit together with computational rules for each node but *no information on the random strings carried by each wire*.

The following holds (see [Rog91]):

Lemma 12 (Indistinguishability) *For any two actors C and D knowing the garbled circuit, if C only knows the garbled version of D's input (and* not *the other random strings carried by D's input wires) garbled circuit evaluation will not disclose more information than if C ran the protocol assuming any random input for D.*

4.7.2 The Wannabe Traveller

For a lighter exposition, and without loss of generality, we present the protocol in the specific setting of an actor W (the Wannabe Traveller) who dispatches an agent in quest of the best offer for a holiday on a tropical island.

The agent visits travel agencies Ag_j, chosen according to some policy that we do not specify here. At each agency, it browses the catalogue and requests the best offer for a holiday matching conditions on the destination, the period, the services, etc., that W requires.

We assume that travel agencies want the privilege of tailoring their offers to the specific client, and therefore prefer that the offer be known only if it is highly likely to be accepted. Moreover each agency does not want that its offer be known to competitors. On the other hand, W does not want to disclose in advance her budget, to avoid that travel agents use it as an information to adjust their offer. Assuming correctness, this leads us to the following definition of the privacy goals we aim at:

Definition 24 (Privacy) *W's agent respects privacy if*

1. *W cannot determine any other offer but the lowest one less than or equal to her budget, if it exists;*

2. *each agency cannot learn W's budget, nor the offer of any other agency.*

4.7.3 The Wannabe-Traveller protocol

Our analysis is focused on the privacy aspects. Therefore we only consider the agent's functionality related to privately (in the sense described above) conveying to W the best offer. Also, we will not model the agent itself as a separate entity. Rather, the agent is represented by a sequence of actions which it repeats identically at each host visited: the collection of sensitive data.

For simplicity, we first consider the case of an agent that visits a single host. W is the source of the agent, and Ag is the host that the agent visits.

In our setting, Ag constructs a garbled circuit that on input $\langle id_1, x\rangle, \langle id_2, y\rangle$, outputs computes $\langle id_{min}, \min(x, y)\rangle$, where $id_{min} = id_1$ if $x = \min(x, y)$ and is id_2 otherwise. We call this function *tagged minimum*. (The description and analysis that follow are absolutely independent of the specific function computed by the garbled circuit.)

Let $input_W$ (resp. $input_{Ag}$) be W's (resp. Ag's) private input.

In order to evaluate the circuit on $\langle id_W, input_W\rangle$ and $\langle id_{Ag}, input_{Ag}\rangle$, W must learn the garbled circuit, the decoding information of the output and the values of her and Ag's garbled inputs. Ag must not learn W's input. In order to transfer to W the garbled inputs corresponding to W's input $\langle id_W, input_W\rangle$, without Ag's learning the value of the input itself, the two parties use an oblivious transfer (OT) protocol [BM89]. (Also see [CCKM00] for the implementation of a one-round oblivious transfer; we assume that W and Ag share a pseudo random generator and a seed.)

Let β be W's committal data for OT, referred to $\langle id_W, input_W\rangle$. Let $GC = gc(\langle id_{Ag}, input_{Ag}\rangle, OT(\beta))$ be the garbled circuit computed by Ag, with Ag's input hardwired into it, and information $OT(\beta)$ attached to it, for obliviously transferring to W her garbled input. We denote by $decode$ the decoding information for the output.

Then the protocol is as follows:

$$\begin{array}{ll} W \longrightarrow Ag: & \beta \\ Ag \longrightarrow W: & \langle GC, decode\rangle \end{array}$$

W computes the single value $tagged\_min(\langle id_{Ag}, input_{Ag}\rangle, \langle id_W, input_W\rangle)$.

Assuming correctness, the protocol guarantees privacy in the sense of Definition 24:

Lemma 13 (Privacy–two parties) *In the honest but curious model, assuming correctness of the protocol, the two party protocol above guarantees privacy in the sense of Definition 24.*

See [CCKM00] for a proof.

In the general case, the agent visits many hosts. We assume, without loss of generality, that the agent travels from W to Ag_1 to Ag_2 and so on, and then back from Ag_n to W.

We generalize and complete the notation that we used for the two-party case. Let $input_W$ be the private input of W, and $input_j$ be the private input of Ag_j. As above, let β be W's commitment to $\langle id_W, input_W\rangle$.

Ag_1 computes a garbled circuit $GC_1 = gc(\langle id_1, input_1\rangle, OT(\beta))$ as in the two-party protocol above. It then forwards $\langle GC_1, decode_1\rangle$ to Ag_2. (The OT data for Alice attached to GC_1 will be forwarded to Alice along with the garbled circuit.)

All other agencies will compute garbled circuits for the tagged minimum function and combine it in cascade one after the other.

For $j > 1$, let $gc_j = gc(\langle id_j, input_j\rangle, transl(decode_{j-1}))$ be the garbled circuit computed by Ag_j, with input $\langle id_j, input_j\rangle$ hardwired to it, and translation information $transl(decode_{j-1})$ for translating the garbled output of the cascade of circuits GC_{j-1} computed by agencies Ag_1 through Ag_{j-1} to a garbled input for gc_j. Then, GC_j is gc_j concatenated to GC_{j-1}. Let $decode_j$ be the instructions for decoding its output.

Ag_j forwards to Ag_{j+1} $\langle GC_j, decode_j\rangle$.

W receives the output of AG_n as if she were an AG_{n+1}, and evaluates it.

The protocol is summarized below ($W \longrightarrow *$ describes W's output of the final value).

$W \longrightarrow Ag_1$:	β	
$Ag_j \longrightarrow Ag_{j+1}$:	$\langle GC_j, decode_j\rangle$	$j = 1,\ldots,n-1$
$Ag_n \longrightarrow W$:	$\langle GC_n, decode_n\rangle$	
$W \longrightarrow *$:	$eval(GC_n, decode_n, \beta)$	

Lemma 14 (Cascade of garbled circuits) *For all $j = 1, \ldots, n$,*

1. *a polynomially bounded actor cannot infer the private inputs of W and $Ag_1, \ldots, Ag_j$ from the garbled circuit $\langle GC_j, decode_j\rangle$;*

2. *moreover, with knowledge of W's (garbled) input, a polynomially bounded actor cannot infer the private inputs of $Ag_1, \ldots, Ag_j$.*

Proof. [Sketch] For $j = 1$, the thesis follows from Lemma 13. For $j > 1$, it can be proven inductively, based on Lemma 12. □

We use TA to prove the following privacy property:

Theorem 4 (Privacy—many parties) *In the honest but curious model, assuming correctness of the protocol, the multi-party protocol above guarantees privacy in the sense of Definition 24.*

The proof of the theorem is given in the sequel as it follows from our TA analysis.

Notation

For convenience of the reader, we summarize the notation that we used to describe the protocol.

- $tagged\_min(\langle t_1, x_1\rangle, \langle t_2, x_2\rangle) = \langle t_{min}, \min(x_1, x_2)\rangle$, where

$$t_{min} = \begin{cases} t_1 & \text{if } x_1 = \min(x_1, x_2) \\ t_2 & \text{otherwise;} \end{cases}$$

- $input_W$ is the private input of W;
- $input_{Ag}$ is the private input of Ag in the two-party protocol;
- $input_j$ is the private input of Ag_j in the multi-party protocol;
- β is W's committal data for OT, referred to $\langle id_W, input_W\rangle$.
- $GC = gc(\langle id_{Ag}, input_{Ag}\rangle, OT(\beta))$ is the garbled circuit computed by Ag, with Ag's input hardwired into it, and information $OT(\beta)$ attached to it, for obliviously transferring to W her garbled input, in the two-party case;
- $decode$ is the decoding information for the output of GC;
- $GC_1 = gc(\langle id_1, input_1\rangle, OT(\beta))$ is the garbled circuit analogous to GC above, computed by AG_1 in the multi-party protocol;
- $decode_j$ is the decoding information for the output of garbled circuit GC_j;
- for $j > 1$, $gc_j = gc(\langle id_j, input_j\rangle, transl(decode_{j-1}))$ is the garbled circuit computed by Ag_j, with input $\langle id_j, input_j\rangle$ hardwired to it, and translation information $transl(decode_{j-1})$ for translating the garbled output of the cascade of circuits computed by agencies Ag_1 through Ag_{j-1} to a garbled input for gc_j;
- GC_j is gc_j concatenated to GC_{j-1};
- $eval(GC_n, decode_n, \beta)$ denotes evaluation of circuit GC_n with decoding information $decode_n$ to interpret the output and additional input β to complete the OT.

4.8 The Wannabe-Traveller protocol modelled by TA

We now show how TA can be used to model the Wannabe Traveller protocol. We model the Wannabe Traveller W by a CA $\mathcal{T}_W$, the set $\{Ag_j \mid 1 \leq j \leq n\}$ of travel agencies by CA $\mathcal{T}_{Ag_1}, \ldots, \mathcal{T}_{Ag_n}$.

Let Input denote the set of pairs $\{\langle id_j, input_j\rangle \mid 1 \leq j \leq n\} \cup \{\langle id_0, input_W\rangle\}$ where $input_j$ (resp. $input_W$) is a string that is private to Ag_j (resp. to W).

Let Computed denote the set of garbled circuits.

Let β be W's OT commitment data. Let Decode denote the set $\{decode_j \mid 0 \leq j \leq n\} \cup \{\beta\}$ where $decode_j$ ($j = 1, \ldots, n$) is the decoding information for the output of circuit GC_j.

Then $\mathcal{T}_{Ag_j}$ uses the function gc : Input $\times$ Decode $\rightarrow$ Computed to compute the garbled circuit gc_j and the function $||$: Computed $\times$ Computed $\rightarrow$ Computed to build up the circuit GC_j consisting of the cascade of garbled circuits gc_1 through gc_j.

Let Result = Input. Then, $\mathcal{T}_W$ evaluates the final result using the function $eval$: Computed $\times$ Decode $\times$ Input $\rightarrow$ Result.

For each $j = 1, \ldots, n$, define $P_j = \langle GC_j, decode_j\rangle$ and $P_0 = \beta$. Then, Messages denotes the set $\{P_j \mid 0 \leq j \leq n\} \cup$ Result.

As in previous sections, we specify TA in the way IOA are commonly defined [Lyn99, LT89].

In all the specifications, we explicitly prohibit loops, *i.e.*, we allow each action to be performed only once. See, for example, the specification of $\mathcal{T}_{Ag_1}$. As soon as $\mathcal{T}_{Ag_1}$ has received P_0, then precondition $P_0 \notin$ received prevents this action to be executed again.

$\mathcal{T}_{Ag_j}$ — Travel Agencies, $j = 1, \ldots, n$

Actions

Inp: $\{P_{j-1}\}$

Out: $\{P_j\}$

Int: $\{Compute_j\}$

States

received, sent $\subseteq$ Messages, computed $\subseteq$ Computed, all initially $\varnothing$

Transitions

P_{j-1}

Pre: $P_{j-1} \notin$ received
Eff: received := received $\cup \{P_{j-1}\}$

$Compute_j$
Pre: $P_{j-1} \in$ received $\wedge\, GC_j \notin$ computed
Eff: computed := computed $\cup \{GC_j\}$

P_j
Pre: $GC_j \in$ computed $\wedge\, P_j \notin$ sent
Eff: sent := sent $\cup \{P_j\}$

The input behaviour $\mathbf{B}^{\Sigma_{inp}}_{\mathcal{T}_{Ag_j}}$ of $\mathcal{T}_{Ag_j}$ $(j = 1, \ldots, n)$ consists of P_{j-1}. When $\mathcal{T}_{Ag_j}$ receives message P_{j-1}, then $\mathcal{T}_{Ag_j}$ is able to perform an internal computation leading to an internal behaviour $\mathbf{B}^{\Sigma_{int}}_{\mathcal{T}_{Ag_j}}$ consisting of $Compute_j$. Finally, the output behaviour $\mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{Ag_j}}$ of $\mathcal{T}_{Ag_j}$, $(j = 1, \ldots, n)$ consists of P_j.

We continue with the specification of $\mathcal{T}_W$. It is capable to output a commitment β to $input_W$. Then, it is capable of receiving as input behaviour the last circuit and the last decoding instructions to evaluate the final result min by starting from what she has received and from $input_W$, by means of function $eval$. Finally, $\mathcal{T}_W$ outputs the final result min.

$\mathcal{T}_W$ — Wannabe Traveller

Actions

Inp: $\{P_n\}$

Out: $\{P_0\} \cup \{min\}$

Int: $\{Eval_{GC_n}\}$

States

received, sent $\subseteq$ Messages, result $\subseteq$ Result, all initially $\varnothing$

Transitions

P_n
Pre: $P_n \notin$ received
Eff: received := received $\cup \{P_n\}$

P_0

Pre: $P_0 \notin \mathsf{sent}$
Eff: $\mathsf{sent} := \mathsf{sent} \cup \{P_0\}$

$Eval_{GC_n}$
Pre: $P_n \in \mathsf{received} \wedge min \notin \mathsf{result}$
Eff: $\mathsf{result} := \mathsf{result} \cup \{min\}$

min
Pre: $min \in \mathsf{result} \wedge min \notin \mathsf{sent}$
Eff: $\mathsf{sent} := \mathsf{sent} \cup \{min\}$

The input behaviour $\mathbf{B}_{\mathcal{T}_W}^{\Sigma_{inp}}$ of $\mathcal{T}_W$ is clearly represented by P_n. When $\mathcal{T}_W$ receives message P_n, then $\mathcal{T}_W$ is able to perform an internal computation leading to an internal behaviour $\mathbf{B}_{\mathcal{T}_W}^{\Sigma_{int}} = Eval_{GC_n}$. Finally, the output behaviour $\mathbf{B}_{\mathcal{T}_W}^{\Sigma_{out}}$ is $\{P_0min, minP_0\}$.

Now, we enforce maximal synchronization between the traveller and the agencies. Thus, the max-ai TA over $\{\mathcal{T}_W, \mathcal{T}_{Ag_j} \mid 1 \leq j \leq n\}$, denoted by $\mathcal{T}_{WT}$, is defined as

$$\mathcal{T}_{WT} = |||\,\{\mathcal{T}_W, \mathcal{T}_{Ag_j} \mid 1 \leq j \leq n\},$$

which formalizes the Wannabe Traveller protocol. From the way CA are composed, the resulting team has no input actions, while it has the union of the output (internal) actions of $\mathcal{T}_W$ and the $\mathcal{T}_{Ag_j}$'s as its output (internal) actions.

4.8.1 Privacy

In this section, we show, through the use of TA, that W's agent respects privacy, in the sense of Definition 24, in the multiparty case ($n > 1$).

We abstract from the syntax details concerning the operations according to which messages can be manipulated, but, as already seen in the previous sections, we assume the presence of an inference system (defined by a derivation operator $\vdash$) that implements these operations.

We restrict the initial knowledge of an automaton A to be bound to a specified set of messages ϕ_A. This informally means that the automaton should be able to produce, by means of only its internal functioning, at most the messages contained in $\mathcal{D}(\phi_A)$. More specifically, when considered as a stand-alone component, the automaton can only execute output actions belonging to $\mathcal{D}(\phi_A)$.

The initial knowledge can be increased to the set ϕ'_A during the execution of the protocol by the messages the automaton receives. Accordingly, the automaton knowledge becomes at most $\mathcal{D}(\phi'_A)$.

We use this notion of knowledge to model privacy. (In order to restrict in a realistic way the inference power of an automaton, we assume, as usual, polynomial boundedness.)

Throughout the analysis, we abstract from the internal computations of the single automata. This is justified by the following: since we are interested in privacy properties, we care about the information flow between the principals, rather than about their internal computations. Thus, in the following we restrict our survey to analyze external actions of our system.

First, we show that W cannot determine any other offer but the lowest one less than or equal to her budget, if it exists (Definition 24(i)).

We must analyze how the knowledge of W is altered in the course of protocol execution. We want to highlight the interactions of W with the rest of the system. Therefore, since we choose to take W's standpoint, communications between agencies, and any distinction among them, are of no interest. So, we combine agencies into a unique block that interacts with W in a way that is indistinguishable from the original system.

We obtain this by defining $\mathcal{T}_{\overline{W}}$ as the max-ai TA over $\{\mathcal{T}_{Ag_j} \mid 1 \leq j \leq n\}$ that is obtained after hiding the actions

$$\Sigma_{com} = \bigcup_{j=1}^{n-1} \Sigma_{com}^{j},$$

i.e., all messages that the travel agencies exchange with each other:

$$\mathcal{T}_{\overline{W}} = \text{hide}_{\Sigma_{com}}(\,|||\,\{\mathcal{T}_{Ag_1}, \ldots, \mathcal{T}_{Ag_n}\})$$

Thus, $\mathcal{T}_{\overline{W}}$ appears as a black box, with some input and output actions it will use to interact with the environment. In our setting W plays the role of the environment. Intuitively, this reflects the nature of the protocol itself: W delegates to its agent the choice of the agencies to visit, and does not need to (and cannot) know details of the agent's transactions with them.

Proposition 12

$$\mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{\overline{W}}} = \mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{Ag_n}}.$$

Proof. The equality follows from the construction of $\mathcal{T}_{\overline{W}}$. □

We can now use Proposition 12 to prove the part of Theorem 4 relative to Definition 24(i):

Proof of Theorem 4 (part 1):

By construction, the initial knowledge of $\mathcal{T}_W$ is bound to $\phi_W = \{\beta, input_W\}$. By definition of the automaton knowledge, the only way in which $\mathcal{T}_W$ can significantly increase its knowledge is by performing input actions. To correctly model the protocol we must impose:

$$\mathbf{B}_{\mathcal{T}_W}^{\Sigma_{inp}} = \mathbf{B}_{\mathcal{T}_{Ag_n}}^{\Sigma_{out}},$$

and therefore, by Proposition 12

$$\mathbf{B}_{\mathcal{T}_W}^{\Sigma_{inp}} = \mathbf{B}_{\mathcal{T}_{\overline{W}}}^{\Sigma_{out}}. \tag{4.3}$$

From the way $\mathcal{T}_{Ag_n}$ is composed, it follows that $\Sigma_{out}^{\mathcal{T}_{Ag_n}} = \{\langle GC_n, decode_n\rangle\}$. From Section 4.8, it follows that $\langle GC_n, decode_n\rangle$ will be executed, and it will be executed only once. Thus, $\mathbf{B}_{\mathcal{T}_{Ag_n}}^{\Sigma_{out}} = \langle GC_n, decode_n\rangle$.

The latter, Proposition 12 and Equation (4.3) imply that the knowledge of $\mathcal{T}_W$ becomes at most $\mathcal{D}(\phi'_W)$, with $\phi'_W = \{\beta, input_W, \langle GC_n, decode_n\rangle\}$. Then, by Lemma 14, and since we are assuming correctness (*i.e.,* GC_n is exactly the garbled circuit computing the (tagged) minimum among all of the private inputs), we conclude that, if W is polynomially bounded, the privacy property of Definition 24(i) holds.

The proof of the second privacy property (Definition 24(ii)) is very similar.

This time we take the standpoint of Ag_j, for any fixed $j \in \{1, \ldots, n\}$. Again we view the rest of the system as a unique block that interacts with Ag_j in a way that is indistinguishable from the way that the collection of the single actors interact with it. To this end we build the following TA:

$$\mathcal{T}_{\overline{Ag_j}} = \text{hide}_{\mathcal{C}}(\,|||\,\{\mathcal{T}_W, \mathcal{T}_{Ag_1}, \ldots, \mathcal{T}_{Ag_{j-1}}, \mathcal{T}_{Ag_{j+1}}, \ldots, \mathcal{T}_{Ag_n}\})$$

where

$$\mathcal{C} = \bigcup_{i=1}^{j-2} \Sigma_{com}^i \cup \bigcup_{i=j+1}^{n} \Sigma_{com}^i \cup \Sigma_{com}^W \cup \Sigma_{out}^W.$$

We prove for $\mathcal{T}_{\overline{Ag_j}}$ a result analogous to Proposition 12:

Proposition 13

$$\mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{\overline{Ag_j}}} = \mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{Ag_{j-1}}},\ \textit{if}\ j > 1,$$
$$\mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{\overline{Ag_j}}} = \mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_W},\ \textit{if}\ j = 1.$$

Proof. The equalities follow from the construction of $\mathcal{T}_{\overline{Ag_j}}$. □

We can now complete the proof of Theorem 4:

Proof of Theorem 4 (part 2):

By construction, the initial knowledge of $\mathcal{T}_{Ag_j}$ is bound to $\phi_{Ag_j} = \{input_j\}$. By definition of the automaton knowledge, the only way in which $\mathcal{T}_{Ag_j}$ can significantly increase its knowledge is by performing input actions. To correctly model the protocol we must impose:

$$\mathbf{B}^{\Sigma_{inp}}_{\mathcal{T}_{Ag_j}} = \mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{Ag_{j-1}}},\ \text{if}\ j > 1,$$

and

$$\mathbf{B}^{\Sigma_{inp}}_{\mathcal{T}_{Ag_j}} = \mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_W},\ \text{if}\ j = 1.$$

Therefore, by Proposition 13

$$\mathbf{B}^{\Sigma_{inp}}_{\mathcal{T}_{Ag_j}} = \mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{\overline{Ag_j}}}. \tag{4.4}$$

From the way $\mathcal{T}_{Ag_{j-1}}$ is composed, it follows that $\Sigma^{\mathcal{T}_{Ag_{j-1}}}_{out} = \{\langle GC_{j-1}, decode_{j-1}\rangle\}$. From Section 4.8, it follows that $\langle GC_{j-1}, decode_{j-1}\rangle$ will be executed, and it will be executed only once. Thus, $\mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_{Ag_{j-1}}} = \langle GC_{j-1}, decode_{j-1}\rangle$. From similar arguments, it follows that $\mathbf{B}^{\Sigma_{out}}_{\mathcal{T}_W} = \beta$.

The latter, Proposition 13 and Equation (4.4) imply that the knowledge of $\mathcal{T}_{Ag_j}$ becomes at most $\mathcal{D}(\phi'_{Ag_j})$, with

$$\phi'_{Ag_j} = \begin{cases} \{input_{Ag_j}, \langle GC_{j-1}, decode_{j-1}\rangle\} & \text{if } j > 1 \\ \{input_{Ag_j}, \beta\} & \text{if } j = 1. \end{cases}$$

Then, by Lemma 14 (if $j > 1$) or Lemma 13 (if $j = 1$) and since we are assuming correctness, we conclude that, if all actors are polynomially bounded, the privacy property of Definition 24(ii) holds.

This concludes the proof of Theorem 4.

Chapter 5

The Digital Certificate Chapter

Formal models and analysis of secure procedures for certificate delivery

5.1 Introduction

ANALYSIS TOOLS. In this chapter, a tool is presented for the automated, finite-state verification of security properties in communication protocols. In particular, the feasibility of the methodology is shown through two case studies, both inherited from real applications that were born to manage the secure delivery of digital certificates over computer networks.

However, our goal is not only focusing on the benefits of the tool. Indeed, from the days it was first theoretically thought and practically developed, [MM99], other formal tools for finite-state verification, involving also stochastic and timed aspects, either have been developed or they have been further optimized through successive updates. Their development has been so far concentrated on two formalisms, process algebras and finite state machine models. Each of them has been, and currently is, largely exploited. Here, we cite and briefly discuss the main features of two general-purpose verification tools, enriched with support for cryptographic primitives, that are not the most recent, but that have been extensively used also to analyze large case studies. They are Murϕ and FDR, respectively [SS98] and [Low96].

FDR is a refinement model checker for the process algebra CSP [RSG$^+$00]. The basic idea is to model the correct behavior of the system as a process algebra

agent, and the protocol specification as another process algebra agent. Then, the tool can check if the specification is a refinement of its correct behaviour.

Murϕ is a finite-state machine verification tool. Its verification method is to use state enumeration with state assertion checking. It is an efficient brute-force reachability analyzer.

In both of them (as well as within other frameworks for finite-state verification), a specification is run exhaustively, and in a specific environment. Sometimes, the modeling process helps in finding bugs, even before running the specification by the tool itself. The motivations should be found in the fact that the modeler is forced to think about the behavior of the protocol participants and the intruder in a much more detailed manner than when writing a protocol specification in a non-formal manner. On the other hand, in the past years troubles with lack of crypto-specification/verification education came up. People designing cryptographic protocols may lack expertise, augmenting the risks for a bug-design version of a final product. Automating the verification process has been proved to be really efficient in finding significant attacks, failures as well as weaknesses, to protocols considered correct even for many years (we refer to the introduction of this thesis, where the paradigmatic example of the Needham-Schroeder Public Key protocol has been discussed).

Here, we get onto the previous discussion, for underlining that the measure of the right trade-off between i) the need for an automated verification methodology (without which some systems would be honestly unfeasible to be checked), and ii) the need for formalizing the model for a system (and the goals that that system aims to) will be searched throughout the chapter.

CASE STUDIES. In the following, we consider (part of) the OpenCA software and (part of) the Simple Certificate Enrollment Protocol (SCEP) as two case studies.

OpenCA Labs (http://www.openca.org) is "an open organization aimed to provide a framework for Public Key Infrastructures studying and development of related projects". In particular, the OpenCA developers aim at implementing open source code to easily setup and manage Certification Authorities (CAs). Consequently, the project provides specification for the whole lifetime of digital certificates, from their emission, through their maintenance, to their expiration. CAs based on OpenCA are really distributed, since there are distinct servers to manage certificates. Furthermore, certificate requests and their validation are performed over the Internet.

SCEP is the evolution of some specification developed by Verisign Inc. and

Cisco Systems and it is commercially available in both client and CA implementations. It gives specifications for digital certificate enrollment, access and revocation, for certificates and CRL queries. In particular, SCEP was developed for the distribution of digital certificates to network devices such as routers and gateways (it is indeed its peculiarity, with respect to implementations like OpenCA, to release certificates to machines, rather than to individuals). Our study is based on the SCEP Internet Draft, [LMMN05] (that should be considered a work in progress, as mentioned by the same authors of the draft).

CONTRIBUTIONS. The chapter offers the following contributions.

- We formally model and analyze (part of) the OpenCA source code and (part of) the SCEP protocol. The results of the analysis are here reported.

  With regard to the OpenCA enrollment procedure, we anticipate that it is correct (at least, at conceptual level). Nevertheless, we focus our attention on the leakage of some sensitive data stored in an on line server. If a malicious adversary discovers these sensitive data, it can force the CA to issue erroneous certificates, in which the public key of a honest user is not tied to the identity of the user itself.

  With regard to the security properties listed in the SCEP draft, we did not find attacks, at least within an analysis scenario consisting of a finite number of participants. While this does not suffice to ensure the absolute security of the protocol in all circumstances, it does enhance the reliability of SCEP. However, we notice a vulnerability concerning the emission of two digital certificates with the same subject name/public key binding.

- The application of automated verification tools is useful to better understand how certain mechanisms and checks ensure certain security features of communication protocols. This might be useful in revealing causes of omitted or erroneously implemented security checks. It might be also useful to render more comprehensive for those less expert some statements written in technical documents, where often it is asserted that a security check is necessary but rarely is the reason given. In order to understand the reason, we simply omit the security check in the description of the protocol, run the verification tool and wait for the result of the analysis. This methodology is particularly useful to study security protocols in a formal way by suitably changing the protocol description in order to simulate possible faults and check the relative effect (as is common in so-called methodology "Failure Model and Effect Analysis" adopted in software engineering, see, *e.g.*,

http://www.fmeainfocentre.com/ and [CG01]). In the future, it would be really appealing to systematically create such case studies in the security protocol analysis framework as done in [CG01] for safety in critical systems.

Finally, to the best of our knowledge, this is the first attempt to give a formal description of some security procedures from OpenCA and SCEP.

We underline that we do not advocate the cause of our tool more than the one of other existing tools. Being this tool developed within our research team, it is more appealing and practical for us to use it. Nevertheless, it would be interesting to compare the results obtained by using our tool with the ones obtaining using other formal instruments.

SUMMARY. The chapter is organized as follows. In Section 5.2, we sketch the methodology of our approach in the security analysis of protocols with finite state behaviour. In Section 5.3, we describe the structures of messages exchanged between parties during the OpenCA enrollment phase. Section 5.4 reports the results we have obtained upon analyzing the OpenCA enrollment phase. In Section 5.5, we describe the structures of messages exchanged between parties during the SCEP enrollment phase. Section 5.6 highlights the motivations for the need of some forms of correctness checks in SCEP specifications: without them, the protocol would be vulnerable to attack by adversaries.

5.2 Analysis approach

This chapter inherits the analysis approach fully illustrated in [Mar03]. Briefly, it was spurred by the observation that security protocols can be described as systems with some component following an unspecified behaviour. This may depend on several factors, *e.g.*, one can simply be unable to predict the whole behaviour of a component within a system. Whatever the not specified term is, it would be appealing that the resulting system works properly (*e.g.*, satisfies a certain property).

As already discussed, given the sensitive nature of a cryptographic protocol, one can imagine the presence of a hostile adversary trying to interfere with the normal execution of the protocol in order to achieve some advantage. Due to the unpredictable behaviour of this adversary, this can be seen as an unspecified component of the system under investigation. When considering formal languages for the description of concurrent systems, such as CCS [Mil89] or CSP [RSG+00],

a concurrent system S with, for example, two specified components A and B and a third unspecified one, can be described as $S = A \mid B \mid (\_)$, where $\mid$ is the parallel composition operator. The hole takes into account the presence of the adversary, whose behaviour is unpredictable.

Starting from these premises, to formally verify a security property of a specification S implies the verification of the property over S with respect to every possible hole (*i.e.*, every possible adversary behaviour). The problem has been discussed in details in [Mar03], and solved by extending the partial model checking techniques of concurrent systems, [And95].

A system specification is here described through the operational language called Crypto-CCS (presented in Chapter 2). The adversary acts according to a Dolev-Yao model, [DY83], and it follows a set of message manipulating rules, that are used, *e.g.*, to model cryptographic functions like encryption and decryption. Like the honest participants, it is able to send and receive messages over a set of channels. Also, it can intercept and forge messages. In part, it can derive new messages from the set of messages that it knows at the beginning of the computation. This set is called the intruder's *initial* knowledge. On the other hand, new messages are derived from the intercepted ones, obtained after the beginning of the computation.

Encryption is opaque, *i.e.*, a message encrypted with the public key of *i* cannot be decrypted by anyone but the person who knows the correspondent private key (unless the decryption key is compromised). The adversary can intercept an encrypted message and it can replay the message later, but the structure of the message is not accessible, *i.e.*, the adversary cannot split the encrypted message unless he knows the decryption key.

The intruder's knowledge grows as the computation evolves. The analysis is over that knowledge, *i.e.*, it is checked if, at a certain point of the computation, that knowledge satisfies a predicate involving a security property. In case of a positive result, this consists of a report of the attack with respect to that property.

The development of the theory has lead to the implementation of a partial model checker, namely the Partial Model Checking Security Analyzer (for short, PAMOCHSA), through which it is possible to analyze distributed systems. In the following, only systems with finite computations will be investigated. This is possible since: 1) we do not allow recursion within Crypto-CCS; 2) the messages are of a fixed structure (due to the fact that they are typed, see Appendix C); 3) a finite number of parties and sessions running the protocol is considered; 4) even if the adversary is allowed to generate fresh messages, their structure is subject to

the same type constraints above mentioned.

It is worth noticing that, though maintaining the analysis over a finite number of parties and sessions, the absence of attacks over a particular system running the protocol does not guarantee that there are no attacks on larger systems running the same protocol. However, [Low98, Sto98] show how, under some assumptions, the correctness of a protocol with an unbounded number of parties and sessions can be inferred from the correctness of the same protocol with finite parties and sessions. Hopefully, mixing the approaches may contribute to the development of a fully automated analysis.

The PAMOCHSA tool needs the following set of inputs: the protocol specification; the security property to be checked; the initial knowledge of the intruder. An example input file and the input language used to write the input file are given in Appendix C.

In the following, the OpenCA and the SCEP enrollment phase will be informally described, by using the standard notation with which security protocols are usually described in the literature. The notation has already been given in Chapter 2.

We remind the interested reader that the functional language used when developing the theory (*i.e.*, Crypto-CCS) and the specification input language for PAMOCHSA are presented, respectively, in Chapter 2 and in Appendix C.

5.3 The OpenCA enrollment phase

OpenCA Labs is an open organization aimed to provide a framework for public key infrastructures studying and development (http://www.openca.org). An open source code has been developed, and it is being maintained, by the OpenCA developers for the setup and management of Certification Authorities. This software is actually a front end to the CA facilities offered by the OpenSSL software (http://www.openssl.org). A formalization of the enrollment procedure of OpenCA will follow.

The following entities are involved in the procedure:

- User (U), requesting a certificate.
- Enrollment Server (ES): a web server used by the users to make certificate requests, import CA Certificate, import requested certificates and import other users' certs. In the investigated implementation, this server is activated

on the same machine of the RA Server (they are indeed the same). All the interactions between ES and the user are through SSL, [FKK96] (client authentication is not required).

- Local Registration Authority Operator (LRA): trusted operators verifying the correctness of a certificate request. In the investigated implementation, only one operator will be considered.

- Registration Authority Server (RA): the web server to which LRA connects, in order to approve certificate requests. Web connections between LRA and RA are secured through SSL.

- Certification Authority Server (CA): the server where the private key of the Certification Authority is kept. Actually, CA issues certificates. For security needs, this is an off-line server, disconnected by any network. All file transfers (Requests/Certificates/etc..) with other computers get executed via removable support, *e.g.*, floppies.

Hereafter, a correctly issued certificate (where the name of the user is tied to its public key) is formalized through the name of the user and its public key, both signed by CA's private key, *i.e.*, $\{name_U, pk_U\}_{pk_{ca}^{-1}}$. An abstraction of a digital certificate is here considered, not involving fields like the validity period. Given that the analysis in Section 5.4 involves the verification of a correct correspondence between an identity and a public key, and it does not involve temporal validity issues, the structure of the certificate has been here simplified for the sake of readability.

Fig. 5.1 and Fig. 5.2 give a pictorial representation of the enrollment procedure. Detailed explanation on the use of public keys pk_{c1}, pk_{c2} and pk_{c3} are given in Subsection 5.3.1. Here, we briefly anticipate that they serve as a modeling trick to give, at least, confidentiality to the content of the communication.

1. U connects to ES and sends a certificate request consisting of its name $name_U$, a newly created random number pin_U, the public key to be certified pk_U and the so-called Netscape SPKAC, *i.e.*, its public key plus a newly created random number n_U, both signed with U's private key pk_U^{-1}. SPKAC acts as a Proof of Possession (POP), attesting that U, that is requesting to link its name to a public key, also owns the corresponding private key. (Only someone who knows the private key can create that SPKAC.) n_U is inserted to prevent replay attacks. It is up to the recipient of the message to

1	$U \mapsto ES$	:	$\{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}}$
2	$U \mapsto LRA$	:	$\{\{name_U\}_{pk_{Gov}^{-1}}, pin_U\}_{pk_{c2}}$
3.1	$LRA \mapsto RA$	:	$\{name_U, pin_U\}_{pk_{c3}}$
3.2	$RA \mapsto LRA$	:	$\{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c3}}$
3.3	$LRA \mapsto RA$	:	$\{\{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{LRA}^{-1}}\}_{pk_{c3}}$
4	$RA \mapsto CA$	:	$\{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{LRA}^{-1}}$
5	$CA \mapsto RA$	:	$\{name_U, pk_U\}_{pk_{ca}^{-1}}$
6	$ES \mapsto U$	:	$\{name_U, pk_U\}_{pk_{ca}^{-1}}$

Figure 5.1: The OpenCA enrollment procedure.

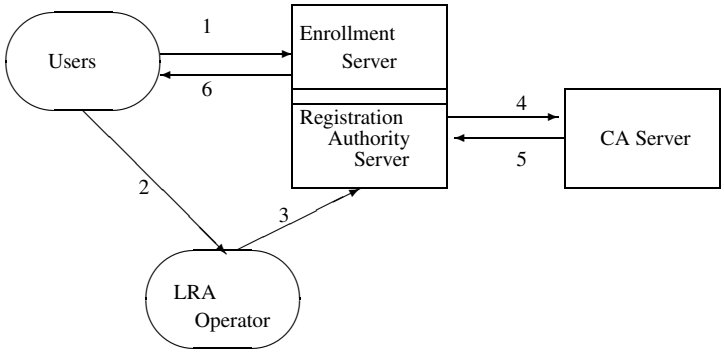

Figure 5.2: Graphical representation of the OpenCA enrollment procedure.

store n_U in order to detect later replays. In the OpenCA specification, this message is sent across SSL. For our rendering of the transmission over SSL the reader is invited to read the following Subsection 5.3.1, *OpenCA model*.

2. U personally reaches the LRA Operator to identify himself by means of a valid paper, *e.g.*, an identity card, and by showing the same pin already present in the formulated request.

   The identity card is here represented by $name_U$ signed by the private key of the Government, *i.e.*, $\{name_U\}_{pk_{Gov}^{-1}}$. It is assumed that it cannot be forged nor leaked.

3. LRA connects to RA and approves the request corresponding to $name_U$ and pin_U. In particular, LRA transmits to RA a query regarding a particular pair identity-pin (step 3.1). Then, RA sends back the request received in step 1 (step 3.2) (corresponding to that pair identity-pin). Finally, LRA sends the request back signed with its private key pk_{LRA}^{-1} (step 3.3). Signing the request means that LRA has personally identified who is requesting the certificate and that there is a correspondence between name and pin received in step 2 and what received in step 3.2. All these three steps are through SSL. Again, the reader is referred to the next subsection for details.

4. The request is exported from RA to CA (through removable media).

5. Before issuing the certificate, CA operator must verify the presence of a correct LRA signature over the request. Then, CA operator checks the correctness of SPKAC (by applying pk_U that is present in message 4). If the checks are satisfied, the certificate is issued and exported into RA (through removable media).

6. Finally, U connects to ES and gets its certificate. (Note that in the investigated implementation ES and RA coincide.)

5.3.1 OpenCA model

All the interactions with the Certification Authority Server (steps 4 and 5) are via removable media through trusted operators. No adversary may intercept or listened to the exchanged messages (at least with high probability).

As far as the other steps are concerned, some modeling assumptions of the OpenCA enrollment are hereafter listed.

- In the OpenCA implementation, steps 1, 3.1, 3.2, 3.3 are performed through SSL connections. That means that data sent over the network are kept confidential (*i.e.*, only the intended recipients can understand the meaning of those messages), and that the server identity is somehow authenticated to prevent server spoofing.

  Step 1 requires no client authentication, *i.e.*, only the user must be assured that the other party is actually the enrollment server to which it is to request a certificate.

  On the contrary, steps 3.1, 3.2 and 3.3 require both client (LRA) and server (RA) authentication, *i.e.*, both RA and LRA must be assured about the identity of the interlocutor.

  To completely model the overall architecture, the SSL protocol should be encoded within the formalization. To avoid the entire encoding, some confidential channels established between the interested parties are assumed. In particular, if one supposes that all possible users (and even the adversary) know a public key pk_{c1} and that only ES knows the corresponding private key pk_{c1}^{-1}, encrypting requests in step 1 with pk_{c1} guarantee their confidentiality. One may observe that the same does not guarantee server authentication. Given that the analysis is here focused on the right correspondence between a public key and a name in a issued certificate, and since it is not concerned with specific SSL properties, the claim is that one can afford to maintain the formalization to a lower level of granularity with respect to the SSL protocol. Moreover, if no attack is discovered with a *weaker* model, one may optimistically imagine that no attack would be discovered with a stronger representation (with respect to the same analyzed properties). Thus, we assume that everybody knows pk_{c1} and only ES knows pk_{c1}^{-1}.

  The goal of confidentiality is similarly modelled in steps 3.1, 3.2 and 3.3, with the specification that the key pair pk_{c3}, pk_{c3}^{-1} is only known by LRA and RA.

- The formal language specifying the protocol is tailored for describing software systems where all the interactions are made by means of communications. Thus, instead of considering that U physically reaches LRA, a sending operation has been modeled in step 2. Furthermore, to maintain the confidentiality that the physical action of showing the data to an operator should guarantee, a confidential channel has been established by means of

pk_{c2}. pk_{c2} is known by all the users (plus the adversary), but only LRA knows pk_{c2}^{-1}.

- In Subsection 5.4.1 two analyses of the enrollment procedure will be carried out by assuming that an adversary gathers information that should be secret, *i.e.*, pin_U and SPKAC, respectively. In the respective specification files, the leakage of these information is modeled by sending them on a public channel. Alternatively, it could also be modeled by including them in the initial knowledge of the adversary. Both the formalizations do not alter the result of the analyses. The first choice differs in the sense that one can decide at which point of the computation the adversary will know the secret.

5.4 OpenCA analysis

An analysis of the correct issuance of a digital certificate is here performed, with respect to an active adversary that tries to interfere with the enrollment procedure described in Section 5.3. The adversary is able to listen, intercept and forge communication between honest participants. An incorrect certificate is here intended as a certificate that testifies the association between a public key and the owner of a private key that does not correspond to that public key.

In particular, two possible misbehaviours are considered for the analysis. Within the enrollment procedure described in Section 5.3, it is investigated if it is possible

1. to issue certificates in which the name of an user is tied to the public key whose correspondent private key is only known by another user;

2. to issue certificates in which the public key of an user, that also knows the correspondent private key, is tied to the name of another one (that does not know that private key).

If one of the two alternatives occur, it means that the enrollment procedure leads to the issuance of certificates in which there is not a correct correspondance between the owner of the certificate and the certified public key.

The occurrence of the first alternative could cause a *responsibility* attack, *i.e.*, someone could sign some documents and makes another user responsible for that signature. The failure of the second property could cause a *credit* attack, *i.e.*, someone could claim credit for the origin of a document signed by another user. It is worth noticing that Abadi has formally discussed the properties of responsibility and credit in [Aba98]. Also, [GMP04] further investigates these topics and

an attempt is given to formalize, within a theoretical framework, some of the examples discussed by Abadi in [Aba98].

First, the initial knowledge of the adversary is set to the set of public messages that it knows at the beginning of the computation, *e.g.*, the names and the public keys of the other participants, its public/private key pair, its identity card.

Thus, the input given to the tool is, informally, as follows (details in Appendix C):

- **Specification file**: OpenCA
- **Formula**:
  $\{name_U, pk_X\}_{pk_{ca}^{-1}}$ *or* $\{name_X, pk_U\}_{pk_{ca}^{-1}}$
- **Initial knowledge**:
  $name_X, pin_X, pk_X, pk_X^{-1}, \{name_X\}_{pk_{Gov}^{-1}}, pk_U, name_U, pk_{c1}, pk_{c2}$.
- The result is **No attack found**.

The specification file is written by following the input language shown in Appendix C. The same appendix contains some excerpts of that specification file, as well as another excerpt from the SCEP specification, that will be discussed and analysed in the following sections.

By requiring the analysis over that particular formula, a computation is searched such that, at the end of such a computation, the adversary knows either message $\{name_U, pk_X\}_{pk_{ca}^{-1}}$ or message $\{name_X, pk_U\}_{pk_{ca}^{-1}}$ (*i.e.*, certificates attesting a wrong association between a public key and its owner). The result of the analysis gives that such a computation does not exist. Thus, the procedure is correct, with respect to the investigated properties, and at least at conceptual level.

5.4.1 Some attacks on the RA server

In the previous part it has been shown that the enrollment procedure of OpenCA leads to the emission of correct certificates. However, under some circumstances, it is possible to force the emission of incorrect certificates. This can happen without breaking CA, but by performing some preliminary attacks on RA. Indeed, the leakage of some confidential information, recorded into RA, may lead to an incorrect certificate issue.

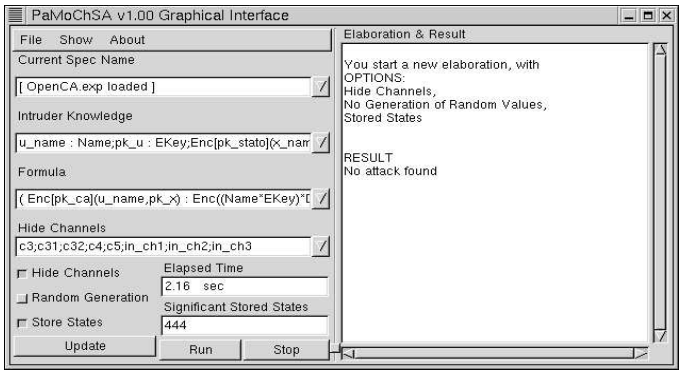

Figure 5.3: Graphical interface of PAMOCHSA

As is common in security analysis, the protocol is investigated by considering that the adversary *magically* gathers some information.

In particular, here it is supposed that X obtains pin_U and SPKAC. Note that this information can be obtained by performing a direct attack on the RA server, which is an on-line machine and it is more vulnerable to attacks than the off-line CA.

Two attacks can occur, depending on the leaked secret:

- X knows pin_U: then, X can associate its public key to the user's identity.
- X knows SPKAC: then, X can associate its name to the public key of user[1].

First attack.

X knows pin_U. The input of PAMOCHSA is:

- **Specification file**: OpenCA_known_pin.
- **Formula**: $\{name_U, pk_X\}_{pk_{ca}^{-1}}$.
- **Initial knowledge**: same as before
- The result is **Attack found**

[1]This holds because of an incomplete management of SPKAC in the investigated version of OpenCA, see Section 5.4.2 for details.

Indeed, X is able to force the issuing of a certificate where its public key is tied to the name of U. Informally, the attack consists of the following steps (see also Fig. 5.4) (In the picture, step i indicates the normal execution of the protocol by the honest participants, as illustrated in Fig. 5.1, whereas step i_X indicates that step i is performed by the intruder):

- Item 1. U connects to ES, as usual.
- Item 1_X. X connects to ES and sends a certificate request consisting of username $name_U$, pin_U it has previously discovered, its public key pk_X and the so-called Netscape SPKAC, *i.e.*, X's public key plus another nonce n_X, signed by pk_X^{-1}.
- Item 2. U gets in contact with the LRA Operator to prove that the data in the request of step 1 are correct.
- Item 3.1. LRA Operator connects to RA Server and approve the request corresponding to the related username and pin_U. However, now two requests exist, both related to the same name and pin. Thus, it could be possible that the LRA Operator approves the wrong request. (Note that steps 3.2 and 3.3 are not formalized in Fig. 5.4, for the sake of simplicity).
- Steps 4, 5 and 6 are as usual. The final issued certificate ties the identity of the user with X's public key.

The avoidance of this kind of attack should be actuated by enforcing security policies to keep sensitive values (like pin_U) secret.

Second attack

X knows SPKAC. (Once again, it is supposed that the adversary receives this piece of information over a public channel. See Subsection 5.3.1.) The input of PAMOCHSA is:

- **Specification file**: OpenCA_known_SPKAC.
- **Formula** $\{name_X, pk_U\}_{pk_{ca}^{-1}}$.
- **Initial knowledge**: same as before.
- The result is **Attack found**.

1	$U \mapsto ES$	:	$\{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}}$
1_X	$X \mapsto ES$	:	$\{name_U, pk_X, pin_U, \{pk_X, n_X\}_{pk_X^{-1}}\}_{pk_{c1}}$
2	$U \mapsto LRA$	:	$\{\{name_U\}_{pk_{Gov}^{-1}}, pin_U\}_{pk_{c2}}$
3.1	$LRA \mapsto RA$	:	$\{name_U, pin_U\}_{pk_{c3}}$
3.2	$as \quad usual$		
3.3	$as \quad usual$		
4	$RA \mapsto CA$	:	$\{name_U, pk_X, pin_U, \{pk_X, n_X\}_{pk_X^{-1}}\}_{pk_{LRA}^{-1}}$
5	$CA \mapsto RA$	:	$\{name_U, pk_X\}_{pk_{ca}^{-1}}$
6	$ES \mapsto U$	:	$\{name_U, pk_X\}_{pk_{ca}^{-1}}$

Figure 5.4: OpenCA - First attack

Informally, the attack consists of the following steps (see also Figure 5.5):

- Item 1. U connects to ES as usual.
- Item 1_X. X connects to ES and sends its certificate request consisting of $name_X$, a newly created nonce pin_X, the user public key pk_U and the leaked SPKAC of the user.
- Item 2_X. X gets in contact with the LRA operator to prove that the data contained in the request in step 2 are correct. X can physically reach the LRA operator and it can show its identity card.
- Item 3.1. The LRA operator connects to the RA server and it approves the request corresponding to $name_X$ and pin_X.
- Item 4. The certificate requests are exported from the RA server and imported into the CA server, as usual. The CA operator checks if all the requests are signed by the LRA operator and if the Netscape SPKAC is correctly signed. This is done by checking that pk_U is able to verify the signature on SPKAC.
- Item 5. The CA operator issues the certificate and exports it into the RA server.
- Item 6_X. X connects to ES and gets the certificate in which pk_U is tied to X's identity.

1	$U \mapsto ES$	:	$\{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}}$
1_X	$X \mapsto ES$	:	$\{name_X, pk_U, pin_X, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}}$
2_X	$X \mapsto LRA$	:	$\{\{name_X\}_{pk_{Gov}^{-1}}, pin_X\}_{pk_{c2}}$
3.1	$LRA \mapsto RA$	:	$\{name_X, pin_X\}_{pk_{c3}}$
3.2	$as \quad usual$		
3.3	$as \quad usual$		
4	$RA \mapsto CA$	:	$\{name_X, pk_U, pin_X, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{LRA}^{-1}}$
5	$CA \mapsto RA$	:	$\{name_X, pk_U\}_{pk_{ca}^{-1}}$
6_X	$ES \mapsto X$	:	$\{name_X, pk_U\}_{pk_{ca}^{-1}}$

Figure 5.5: OpenCA - Second attack

Prot. descriptions	states	CPU time	Result
OpenCA_NO_attack	444	2.05 sec	NO Attack (see 5.4)
OpenCA_Known_PIN	453	0,25 sec	Attack (see 5.4.1)
OpenCA_Known_SPKAC	350	0.27 sec	Attack (see 5.4.1)

Figure 5.6: OpenCA: results.

5.4.2 A note on the use of SPKAC

The software architecture investigated above is based on OpenCA v.0.2.0 and OpenSSL v.0.9.4. As explicitly mentioned in the OpenSSL documentation, the SPKAC nonce challenge is not used in that implementation. Thus, the second attack can be avoided by simply recording the nonce previously used in a SPKAC structure.

In order to look for a practical evidence of this drawback, a test Certification Authority has been built, based on the 0.9.4 SSL code. Actually, two requests with the same SPKAC have been generated and, consequently, two certificates have been issued, with different user name but same public key. However, if SPKAC is correctly used, a single CA cannot issue such erroneous certificates.

On the other hand, in a more complex PKI infrastructure this is not true anymore. Let the reader consider a PKI infrastructure consisting of a root CA and two sub-CAs, CA_1, and CA_2 (*i.e.*, a hierarchical structure). The root issues the certificates for its sub-CAs, while these directly interact with the users. Let us now suppose that user A discovers the SPKAC of user B. B is also supposed to have previously obtained a valid certificate from CA_2. A can send a correct certificate request to CA_1, by using its name and B's SPKAC. As a consequence, A may obtain a valid certificate where its name is tied to B's public key. Under this scenario the nonce is useless, since CA_1 receives the SPKAC for the first time.

A possible solution could be the insertion in the SPKAC structure of some additional information, by following, *e.g.*, the example of the PKCS#10 structure, that is a standard describing the syntax for making a certificate request. In subsection 5.5.1 the interested reader will find more about the format and usage of this standard.

Here, it is anticipated that, by following the PKCS#10 standard, the user request contains a digital signature (generated with the user's private key) that signs the name of the user, plus some additional attributes. Then, it makes no sense to perform a duplicate request, as the one in step 1_X of Fig. 5.5. Such an attack would not work, simply because the LRA operator is now able to check if the name of the user making the request matches the name in the signed field. Updating the protocol by following the new guidelines, step 2 becomes:

$$X \mapsto ES : \{name_X, pk_U, pin_X, \{name_U, pk_U, attributes\}_{pk_U^{-1}}\}_{pk_{c1}}$$

Thus, X cannot simply reply the request (if not in possess of pk_U^{-1}). Indeed, the devoted operator would find a mismatch between the identity within the signature

($name_U$) and the identity outside the signature ($name_X$).

Upon analyzing the protocol through PAMOCHSA, the analyzed OpenCA version has been updated by the authors of this paper, by allowing only requests *à la* PKCS#10.

5.5 The SCEP enrollment phase

A description of the Simple Certificate Enrollment Procedure (SCEP) is hereafter given.

SCEP is a two-way communication protocol whose goal is the secure issuance of certificates to network devices, such as routers and gateways, using existing technology. Up to today, the last document describing SCEP is an Internet Draft available on Internet at [LMMN05]. The protocol is one of the first to be adopted by numerous vendors because it offers a common method of enrolling (*i.e.*, requesting and receiving digital certificates) from different Certification Authorities.

SCEP supports the following operations:

- CA public key distributions;
- Certificate Enrollment;
- Certificate Revocation;
- Certificate and CRL query.

Hereafter, the attention will be mainly focussed on the enrollment phase. That consists of two main phases:

1. the SCEP client (also called user U), identified by a subject name consisting of the Fully Qualified Domain Name (*e.g.*, alice.somewhere.com), asks for a digital certificate. It composes its certificate request and sends it to a Certification Authority Server (CA), an entity which issues certificates and whose name will be declared in the certificate issuer name field.
2. The Certification Authority tests the correctness of the received request[2]; in case of positive outcome, CA issues the certificate, digitally signs it and sends it to the applicant.

[2]In subsection 5.5.2 it is explained how CA tests the correctness of the request.

The secrecy and integrity of the private key of the Certification Authority must be preserved. (This is why the CA Server should be an off-line machine. In this case, communications between CA and the applicant pass through an on-line Registration Authority Server RA. However, CA Server and RA Server are specified as one entity in the following).

U generates its pair of asymmetric keys with a specific key usage (*i.e.*, only for encryption, only for digital signature, or both). The public key to be certified and the key usage are conveyed to CA through the certificate enrollment request.

5.5.1 User certificate request

After the generation of the keys and having obtained the CA's certificate, necessary to retrieve CA's public key in order to enroll, the user generates its request using PKCS#10 and sends it to the CA exploiting PKCS#7. PKCS#10 and PKCS#7 were issued by RSA Labs and made public and modifiable as with PKCS#i (Public-Key Cryptography Standards). They are "de facto" standards: PKCS#10 describing the syntax for certification requests and PKCS#7 defining formats to represent data with the addition of cryptographic information, *i.e.*, encrypted data or digital signatures. PKCS#7 provides different kinds of formats, like Signed Data (data plus digital signatures), Enveloped Data (encrypted data plus encrypted key by means of RSA), Degenerated Mode (for the distribution of certificates). Briefly, a PKCS#10 request can be formalized as follows:

$$PKCS\#10 := \{name_U, pk_U, pin_U, \{name_U, pk_U, pin_U\}_{pk_U^{-1}}\}$$

where $name_U$ is the Subject Name of the user U (Fully Qualified Domain Name plus IP Address), pk_U is the public key to be certified and pin_U is a secret that associates the subject name to that certificate request[3].

PKCS#10 is completed by adding the digital signature of the 3-tuple Subject Name, Public Key and Pin, using the user's private key. The signature acts as a Proof of Possession (POP), *i.e.*, once CA has verified the signature, it has an evidence that whoever has originated the signature holds the corresponding private key. The key usage is specified in the PKCS#10; in our formalization the key

[3]This pin is used for certificate revocation (currently implemented as a manual process: the user phones a CA Operator asking for revocation of its certificate, the operator replies asking for the pin. If it coincides with the one contained in the PKCS#10 request, the certificate is revoked). The pin can also be used to authenticate the identity of U, as explained in subsection 5.5.2

usage is not explicitly declared and the usage is intended for both encryption and signature.

Upon composing PKCS#10, U builds the Enveloped Data[4], exploiting PKCS#7 technologies:

$$EnvelopedData := \{PKCS\#10\}_{KEY}, \{KEY\}_{pk_{CA}}$$

where KEY is a randomly generated symmetric key. The construction of Enveloped Data provides the encryption of KEY with the public key of the CA, pk_{CA}, so that only CA can retrieve KEY and successfully obtain the PKCS#10 as a clear-text.

To complete the enrollment request, U creates Signed Data, basically as follows:

$$SignedData := EnvelopedData, \{ID, Nonce\}_{pk_U^{-1}}$$

- The aim of the Transaction Identifier ID is to uniquely identify the transaction. ID is the fingerprint of the public key to be certified.
- $Nonce$ is a random number generated by U, and its aim is to prove the freshness of the response from the CA to the user request.

ID, $Nonce$ are sent as "authenticated attributes" of PKCS#7, signed with private key of U. Answers[5] of CA to U's enrollment request can be of three kinds:

1. SUCCESS response: CA successfully issues the requested certificate;
2. PENDING response: CA is configured to act in manual mode. Before the emission, it has to carry out some checks to verify enrollment request correctness;
3. FAILURE response: checks have produced a negative result and CA does not issue the certificate.

When U receives an answer from CA containing a *pending* "status", it can enter into a polling mode, *i.e.*, it can periodically send to CA *Get Cert Initial* messages, pressing for the certificate emission. The structure of a *Get Cert Initial* message

[4]For the sake of readability the structures of Enveloped Data, Signed Data and Get Cert Initial message here are simplified (without, however, affecting the results of our analysis).

[5]These answers contain the same ID and $Nonce$ present in the User Certificate Request.

is substantially the following:

$$GetCertInitial := \{name_U, ID\}_{KEY}, \{KEY\}_{pk_{CA}}, \{Nonce\}_{pk_U^{-1}}$$

5.5.2 Modeling the enrollment procedure

For the sake of clarity, it is worth spending a few words on the user authentication phase. In protocols that use public key cryptography, the association between the public keys and the identities with which they are associated must be authenticated in a secure manner. SCEP provides two authentication methods: a manual one and one based on a pre-shared secret.

In manual mode, once a certificate request has been sent to CA, U has to wait until its identity can be verified using any reliable mechanism, to be performed over channels other than the Internet. This mechanism could be performed, *e.g.*, by personally reaching a devoted operator and directly showing some appropriate credentials, or by delivering such credentials by surface mail or phone. In particular, [LMMN05] suggests that CA generates the fingerprint of the PKCS#10 retrieved from the user request and compares it with the one computed by the user itself. During this period, the state of the whole transaction is set to "PENDING".

Otherwise, CA can choose to act in automatic mode: before any request takes place, CA should distribute a pre-shared secret to the user - the secret is assumed to be unique for each user (the way in which the distribution takes place is subject to the CA policy). When creating an enrollment request, the user will insert the secret in the PKCS#10 (later on we refer to this secret as pin_U). Upon receiving the request, CA should check the correspondence between pin_U and the subject name included in the PKCS#10.

Enrollment procedure with manual user authentication.

The enrollment procedure with manual authentication of the user can be described as follows:

1. U connects to CA and sends enveloped PKCS#10 and authenticated attributes. ID_U is the fingerprint of pk_U. $Nonce_U$ is inserted in the authenticated attributes to prevent replay attacks from the user point of view. In this procedure, every answer from CA to U has to contain the same nonce of the previous message from U to CA.

1	$U \mapsto CA$	:	$\{name_U, pk_U, pin_U, \{....\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
2	$CA \mapsto U$	:	$\{ID_U, Nonce_U, \text{“}pending''\}_{pk_{CA}^{-1}}$
3	$U \mapsto CA$	:	$\{name_U, ID_U\}_{KEY}, \{KEY\}_{pk_{CA}}, \{Nonce1_U\}_{pk_U^{-1}}$
4	$CA \mapsto U$	:	$h\{name_U, pk_U, pin_U, \{....\}_{pk_U^{-1}}\}$
5	$U \mapsto CA$	:	$Comparison : ok/ko$
6	$CA \mapsto U$	:	$\{\{name_U, pk_U\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_U}, \{Nonce1_U\}_{pk_{CA}^{-1}}$

Figure 5.7: SCEP Enrollment Phase – manual mode.

2. CA is configured to manually authenticate the end entity, so it sends a PKCS#7 message back to U containing only authenticated attributes: “status” of transaction set to *pending*, same transaction identifier and same nonce as in the user request.

3. Upon receiving a pending status, U enters into polling mode by periodically sending *Get Cert Initial* messages to CA, until it either receives the certificate, or notification of rejection, or it simply times out. (Here, it is assumed that the user successfully obtains its certificate after the sending of the first *Get Cert Initial*.)

4. Communications over channels 4 and 5 should be intended other than the Internet. (The analysis tool allows these two channels to be hidden from the intruder, in order to simulate a reliable secure communication between CA and all the users). This reliable communication is intended to be by phone or by surface mail. In any case, CA must securely contact U and it must communicate to U the fingerprint of the PKCS#10 received in Message 1 (Message 4). Thereafter, the user can compare the fingerprint with the one computed from its original PKCS#10 (Message 5).

5. U gives a positive, or negative, answer to CA, depending on the result of the comparison.

6. Upon receiving a positive answer, CA issues the certificate. SCEP distributes the certificates by following PKCS#7 Degenerated Mode, enveloped and followed by the same user nonce contained in the previous *Get Cert Initial*.

Enrollment procedure with automatic user authentication.

When a pre-shared secret scheme is used, the enrollment procedure is quite simple (Fig. 5.8). The user authentication is subject to the correspondence between pin_U and $name_U$.

5.6 SCEP analysis

A formal analysis of security protocols turns out to be a useful mechanism to better understand the motivations and the choices for the intrinsic structure of a message, as specified in documents like Internet standards and drafts.
Briefly, the security goals of SCEP are that no adversary can:

1. subvert the public key/identity binding from that intended;

2. discover the identity information in the enrollment request and in the issued certificates;

3. cause the revocation of certificates with any non-negligible probability.

The first and second goals are met through the use of the standards PKCS#7 and PKCS#10 (by exploiting encryption and digital signatures with authenticated public keys). The third goal is met through the use of a challenge password for revocation.

The revocation phase is not a concern to the scope of this paper but rather the phase of enrollment has been considered. The analysis deals both with the enrollment procedure with manual authentication of the user and with the automatic procedure.

When running the tool, a finite number of processes, each of them having a finite behavior, has been considered. Note that, with regard to this scenario, SCEP guarantees the correct emission of certificates (*i.e.*, goals 1 and 2 are achieved).

---

1 $U \mapsto CA$: $\{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
2 $CA \mapsto U$: $\{\{name_U, pk_U\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_U}, \{Nonce_U\}_{pk_{CA}^{-1}}$

Figure 5.8: SCEP Enrollment Phase – automatic mode.

However, the attention will be focused on particular checks suggested in [LMMN05], in order to understand some security mechanisms and the possible consequences of their absence.

In particular, in subsection 5.6.1 the security mechanisms suggested in the Internet Draft to achieve Property 1 are considered. Namely, it is considered which mechanisms are involved and whether without them it would be possible to issue a certificate with a erroneous correspondence between an identity and the public key to be certified.

The possibility of issuing two (or more) certificates with same subject name - public key - key usage binding (whose validity periods overlap) is also taken into account. In subsection 5.6.2 it is discussed why the event should be avoided, what [LMMN05] suggests and what could happen without these suggestions.

5.6.1 Relevance of the user authentication.

Goal 1 in section 5.6 is that no intruder can force CA to issue erroneous certificates in which the public key/identity binding is subverted. Here, a simple analysis is performed, by checking if an intruder could be able to break that goal.

If CA was able to issue certificates in which the name of a legitimate user is tied to a public key provided by an enemy, a so called "*responsibility* attack" could be caused. Indeed, someone could sign something and make another person responsible for that signature.

A specification following the Internet Draft (*i.e.*, that one including the comparison of the PKCS#10 fingerprints) has been checked with the help of the tool. The analysis confirms the correct emission of the certificate.

On the contrary, when no fingerprint comparison takes place, the protocol results vulnerable to a man in the middle attack. The tool automatically unveils the attack. This consists of the following steps (see also Fig. 5.9):

The attack consists of the following steps (see also Fig. 5.9):

1. U connects to CA as in a normal execution. The request is intercepted by X.
2. X sends to CA a certificate request containing the user name.
3. CA's answer contains a pending status.
4. U enters into polling mode. Its *Get Cert Initial* message is intercepted by X.

1	$U \mapsto X(CA)$	:	$\{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
2	$X(U) \mapsto CA$	:	$\{name_U, pk_X, pin_X, \{name_U, pk_X, pin_X\}_{pk_X^{-1}}\}_{KEY_X}, \{KEY_X\}_{pk_{CA}}$, $\{ID_X, Nonce_U\}_{pk_X^{-1}}$
3	$CA \mapsto U$	:	$\{ID_X, Nonce_U, \text{“}pending''\}_{pk_{CA}^{-1}}$
4	$U \mapsto X(CA)$	:	$\{name_U, ID_U\}_{KEY}, \{KEY\}_{pk_{CA}}, \{Nonce1_U\}_{pk_U^{-1}}$
5	$X(U) \mapsto CA$	:	$\{name_U, ID_X\}_{KEY_X}, \{KEY_X\}_{pk_{CA}}, \{Nonce1_U\}_{pk_X^{-1}}$
6	$CA \mapsto U$	:	$\{\{name_U, pk_X\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_X}, \{Nonce1_U\}_{pk_{CA}^{-1}}$

Figure 5.9: No fingerprint comparison.

5. X simulates the polling mode.

6. Something went wrong with the comparison of the fingerprint. It is possible to issue the certificate related to X request.

The absence of the fingerprint comparison could represent either the fact that CA does not contact the user to communicate the received fingerprint (no Message 4 in Fig. 5.7) or the fact that the user itself omits the comparison (no Message 5 in Fig. 5.7).

Note 2 *The particular structure of the messages in SCEP helps U to discover that it actually receives a wrong certificate. Upon receiving CA's answer containing the certificate (message 6), U will not be able to open the envelope, since the symmetric key* $KEY1$ *is encrypted with the incorrect key,* pk_X*. However, X could intercept message 6, so that the user does not receive the certificate. In these circumstances, the user presumably sends to CA a sequence of* GetCertInitial *messages, pressing for the certificate. The possible interceptions of* Get Cert Initial *messages by the intruder and the consequent time out interrupt of U's connection may lead U to realize that something incorrect has happened.*

5.6.2 How to avoid the issuance of two identical certificates

For authenticating the user in automatic enrollment, (Fig. 5.8), it is recommended the use of a pre-shared secret scheme. The pre-shared secret is represented, in the given formalization, by pin_U. As CA receives a request of enrolment, it should verify the correspondence between the identity of the applicant and its secret.

Furthermore, [LMMN05] encourages CAs to enforce the so called *certificate-name uniqueness*: at any time, there will be only one pair of keys for a given subject name and key usage combination[6].

We prefer to distinguish between two kinds of uniqueness. Thus, the above mentioned property will be referred as a *weak uniqueness*, meaning that it is not possible to issue two (or more) valid certificates with the same subject name, same public key and key usage whose validity periods overlap. With respect to the validity period of a certificate, weak uniqueness is in contrast with another property, called *strong uniqueness*, meaning that it is *never* possible to issue two (or more) certificates with the same subject name, same public key and key usage.

To better distinguish between weak and strong uniqueness, let the reader suppose the existence of two certificates with same subject name, same public key and key usage. Then, it could be the case that their validity periods overlap (*e.g.*, the validity period of the first certificate is January 1st, 2005 – January 1st, 2006, whereas the one of the second certificate is July 1st, 2005 – July 1st, 2006). It could be also the case that their validity periods do not overlap at all (*e.g.*, January 1st, 2005 – January 1st, 2006 and July 1st, 2006 – July 1st, 2007). In the first case, both weak and strong uniqueness do not hold. In the last case, it holds weak uniqueness, but not the stronger one.

In [LMMN05], U is allowed to re-use the same request when it times out from polling for a pending request (or when the connection between U and CA goes down for some reason). In any case, the second request should not create a new transaction nor should the second request be rejected. So what should the re-emission lead to? There are some possibilities expressed in the draft:

- If CA has already issued the certificate and the re-emission of the request occurs less than halfway through the validity time of an existing certificate, then CA should realize that something went wrong in the first sending of the certificate. It can possibly re-send the same certificate to the applicant, without issuing another certificate.

- If CA has already issued the certificate, and the re-emission of the request occurs more than halfway through the validity time of an existing certificate,

[6]Uniqueness is not mandatory in the draft, but the draft itself claims that all current SCEP client implementations expect the property. Moreover, the authors of the draft made examples by assuming that uniqueness holds. Thus, the current analysis is performed assuming that the property holds.

this can be interpreted by CA as a renewal request. In that case, CA should have previously revoked the existing certificate[7].

- If CA has not yet issued the certificate, the reception of the same request should be taken by CA as another $GetCertInitial$ message, instead of a request for a new enrollment.

Let the reader consider the hypothesis that CA issued two certificates with the same subject name U, the same public key and key usage. They will differ in the serial number, say $sn1$ and $sn2$, and the respective validity periods will overlap. Suppose also that X was able to force CA to issue the last certificate, so X is conscious of its existence, while U is not. There are multiple reasons for preventing the re-transmission of the same data from creating a second certificate. The most significant reasons, according to the authors of this thesis, are:

- considering a large scale application scenario, the computational cost in generating and signing unused certificates can be high;
- a document digitally signed by U (*i.e.*, by the private key corresponding to U's public key) could be valid longer than expected, because the unexpired certificate would validate the signature;
- U could maliciously extend his own certificate validity even when it is purposely denied the right to a new certificate, *e.g.*, in a corporate environment, an employee might have access to a certain facility but only for a limited time.

Each public key to be certified is strictly connected to the Transaction Identifier ID, *i.e.*, the fingerprint of the public key. To guarantee weak uniqueness, it is assumed that CA records the pair $(name_U, ID_U)$.

A specification expecting weak uniqueness (*i.e.*, a specifications including the record of the pair $(name_U, ID_U)$) has been checked with the help of PAMOCHSA. The analysis confirms that it is not possible to issue two identical certificates whose validity periods overlap.

On the contrary, if CA neglects to record that pair, the protocol results vulnerable to the following attack: the intruder could eavesdrop on a legitimate certificate

[7]Before revoking the existing certificate, CA should presumably contact the user for confirmation, but this is not specified in [LMMN05].

request and simply repeat it later. In this way, X can force CA to issue two identical certificates - apart from the serial number. This replay attack is reported in Fig. 5.10.

1. U connects to CA as usual. Its request is eavesdropped by X.
2. X connects to CA and repeats the initial U's request.
3. CA issues a first valid certificate with serial number sn_1.
4. CA issues a second valid certificate with a different serial number, because it omits checks on crucial fields in the received requests. X is able to intercept the message.

The user itself may unconsciously contribute towards the issuance of two identical certificates. Indeed, if U times out (or the connection to CA goes down for some reason), CA could issue a first certificate but U may not receive anything because of the connection crash (or the time out). Consequently, U is allowed to re-issue the same request and the absence of checks by CA may lead to the issuance of a double certificate.

In the discussion above, the existence of certificates whose validity periods overlap has been investigated. When automatic enrollment is used, weak uniqueness is not enough to protect against replay attacks on expired certificates requests. Indeed, X could eavesdrop on a legitimate certificate request. The automatic procedure will lead to the issuance of a legitimate certificate, say $\{name_U, pk_U\}_{pk_{ca}^{-1}}$. Then, X may send the replay of the request once the legitimate certificate has expired. This can cause a new certificate to be issued with the same subject name - public key binding. The existence of the last certificate may cause a document previously signed by U to be valid longer than expected. It is our opinion that, to avoid this vulnerability, CAs should guarantee strong uniqueness.

1	$U \mapsto CA$	:	$\{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
2	$X(U) \mapsto CA$	:	$\{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
3	$CA \mapsto U$	:	$\{\{name_U, pk_U, sn_1\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_U}, \{Nonce_U\}_{pk_{CA}^{-1}}$
4	$CA \mapsto X(U)$	:	$\{\{name_U, pk_U, sn_2\}_{pk_{CA}^{-1}}\}_{KEY2}, \{KEY2\}_{pk_U}, \{Nonce_U\}_{pk_{CA}^{-1}}$

Figure 5.10: Replay attack.

Chapter 6

Conclusions

In this section, conclusions are given with respect to what has been presented in the body of this thesis. Conclusions related to what presented in the appendixes A and B are left to those appendixes themselves, since, obviously, that material has not been exposed yet.

The main thread of this work has been to study methodologies for the modeling and analysis of security protocols. The investigation has been made by considering cryptography reliable. Instead, we had an eye for formal methods.

Even though the literature offers a lot of interesting work in the area, the field of survey was so wide that we have decided to investigate some aspects not deeply analyzed so far.

Multicast security being a fertile field for computer science and engineering researchers and developers, a subsequent attention was focused on methodologies for certifying the goodness of the developed architectures. Thus, by means of both a process-algebraic and an automata framework, we have modeled real life protocols for signing and protecting digital contents. Also, we have verified some of their properties by means of formal proofs.

Then, our attention was focused on the study of some informal documents giving guidelines for correctly ensuring authentic correspondences between a public key and an identity. A translation of these procedures into more formal specifications has been given. These specifications were given as input to a tool for the automated specification, in order to investigate the effects of omitted (or erroneously implemented) security checks.

Finally, Team Automata have been proved to form a flexible framework for modelling communication between components of distributed and reactive systems [BEKR03, Ell97, Kle03]. Given these premises, they have raised our atten-

tion and we have investigated how to equip them with a framework for treating security aspects.

In the following, we give a more detailed report of the contents and results of the work, chapter by chapter.

6.1 The Multicast chapter

In the chapter dedicated to model multicast protocols and to verify some of their security properties, a compositional analysis has been successfully applied. It has been verified the property of integrity (*i.e.*, a sort of robustness against packet modification). In particular, we have proved that the exchanged multicast data are not modified *en-route*, *i.e.*, in their traveling from one sender to the set of receivers.

The efficiency of modelling (and analysis) has been shown through three case studies, some well-known protocols to sign digital streams. The reader is invited to note that the analysis does not limit itself to check the property of integrity. For example, in the timed case study, the fulfillment of the property of timed integrity is a consequence of the fulfillment of the property of timed secrecy over the keys that are going to be disclosed. We could also have checked explicitly timed secrecy over those keys, with the same proposed machinery.

Systems with an unbounded number of components have been modeled and analyzed.

The choice of the three case studies has not been occasional. The first is considered a pioneer protocol in the field of securing digital streams. However, it suffers from the problem of packet loss, in the sense that, if a packet is missing, the authentication chain is broken and the integrity of the subsequent packets cannot be verified. Several protocols were born with the intent of fighting against this problem. In particular, we have chosen EMSS, in order to model also packet loss. We achieve it through a non-deterministic choice performed at the receiver's side. Finally, also timed issues in wireless environments have been considered. To this aim, a process algebra enriched with timed primitives has been used to model the relevant μTESLA protocol, a time-dependent wireless security protocol. In particular, the analysis framework has been extended by developing a compositional approach for reasoning about security properties that rely on time constraints.

Relevant work related to the verification of digital streams are those in 1) [Arc02], a formal analysis based on theorem proving techniques to analyze a

well known stream authentication protocol (the TESLA protocol [PCST01]) and 2) [BL02], where the authors verify the same protocol by means of model checking techniques.

Notable examples of compositional proof techniques for reasoning about cryptographic protocols may be found in [GT00, BG02]. In [BG02] a compositional proof system for an environment-sensitive bisimulation has been developed. One main difference from ours is that we consider a weak notion of observation where the internal actions are not visible. In [GT00] the concept of disjoint encryption has been developed and the authors were able to perform compositional reasoning both for secrecy and authentication properties. What would be interesting, for the future, it is the study of the relationships between disjoint encryption and our stability assumption. We also note that compositionality is a fundamental issue in static analysis approaches. An ongoing study is indeed the comparison of our approach with the one proposed in [GJ01, GJ02], based on type systems for checking authenticity and integrity properties.

Related work in security protocol verification in a timed setting may be found in [ES00], where a timed variant of the process algebra CSP [RSG$^+$00], namely tock-CSP, is presented. This work presents differences (with respect to ours) with respect to the treatment of the timed operators and the cryptographic modelling. Moreover, [ES00] does not rely on any compositional principle, but it has the appealing advantage that the theory is automatized through the use of PVS ([SOR93]), while ours, up to now, is a *pen and proof* analysis.

6.2 The Team Automata chapter

Team Automata have been equipped with a framework for security analysis by defining a general insecure communication scenario for team automata and by reformulating the GNDC schema, originally developed for process algebras, in terms of team automata. Furthermore, for the given insecure communication scenario, a compositional analysis has been proved and applied.

Firstly, by defining the most general intruder we have been able to avoid the universal quantification in an initial reformulation of the GNDC schema for team automata. Secondly, by defining a compositional analysis strategy for team automata we have shown how some security properties are preserved by composition over an initiator and a responder. Finally, we have used the developed framework to prove that integrity is guaranteed in a case study in which team automata model the EMSS protocol, thus demonstrating the effectiveness of our framework for

security analysis with team automata.

A significant *trait d'union* between the Multicast chapter and the Team Automata chapter is that the same case study, EMSS, has been considered. TA were proved to be a natural formal model for multicast/broadcast communication. On the contrary, some modeling tricks have been adopted to model multicast within a process-algebraic framework. In fact, in the theory of TA, many variants of a concurrent composition operator can be uniformly defined. In particular, we defined a multicast composition operator $\|^J$ so that we were able to model a multicast protocol involving one sender and n copies of a receiver as one-to-J synchronizations between the components of a TA.

Another interesting field of survey has been a proposal for modelling the secure agents of [CCKM00] in the framework of TA. We have investigated a possible way of analyzing privacy properties with TA. To the best of our knowledge this is the first attempt to use TA in order to analyze privacy properties. The insight we gain from the analysis underlines a weakness of this approach to secure agents. Indeed, it emerges in a natural way, that the protocol we study is not agent-oriented in spirit, but it rather offers a means of adding a security layer over agent technologies. Our impression is that such an approach cannot carry very far.

We would like to remark here what a referee stressed about the paper [EP04], in which we describe the analysis of privacy properties of mobile agents. The referee wrote that *the results seem "obviously" true*, appreciatively emphasizing the simplicity of the proofs we had given. This simplicity depends on the fact that TA abstract away from the cryptographic layer and focus on pure communication aspects. On the other hand, the privacy properties of the protocol we analyzed depend mostly on the cryptographic aspects of the construction. We argue that simplicity is a desirable property in the setting of security protocols verification. In our case, we owe it to the fact that we use a communication-based model, that allows us to distinguish which are the sensitive ingredients of a protocol.

6.3 The Digital Certificate chapter

In the chapter dedicated to model and analyze the delivery of a digital certificate, a formalization of the OpenCA and the SCEP enrollment phases has been given. The analysis of part of the security properties of the two procedures has been performed, by means of a software tool. Our survey gives the flavor of the needed trade-off between i) the need for an automated verification (without which some systems would be honestly unfeasible to be checked) and ii) the need for

formalizing in a rigorous way the model of a system and its goals.

Contrary to Chapter 3, this chapter is about a finite-state verification. Thus, a scenario with finite number of processes, each of them with a finite behavior, has been taken into account. With respect to this scenario, we did not find any attack, meaning that all the analyzed properties hold.

However, with regard to OpenCA, it has been shown that, when sensitive data are not accurately stored, the leakage of that data may lead to an incorrect certificate issuance. This can cause attacks on responsibility and credit, *e.g.*, by making a signature to be considered legitimate instead of invalid, or by considering someone responsible for something (s)he has not actually signed. Hence, such an analysis acts as a reminder for correctly implementing security checks, and not only informally specifying them.

With regard to SCEP, when automatic enrollment is used, we found a vulnerability concerning a replay attack on expired certificates requests. To deeply understand certain security mechanisms in SCEP specifications, we purposely omitted particular checks suggested in the draft in some of our experiments. As a consequence, an attack regarding the issuance of certificates with the public key/identity binding subverted and an attack concerning duplicate valid certificates were automatically detected.

The proposed analysis technique is general enough to encompass several classes of security protocols (with finite behaviour). We remark that it turns out to be a valid support for the analysis of RFCs, drafts, codes and commercial products.

Bibliography

[AB99] G. Anastasi and A. Bartoli. Group multicast in distributed mobile systems with unreliable wireless network. In *Proc. SRDS'99*. IEEE, 1999.

[Aba98] M. Abadi. Two facets of authentication. In *Proc. CSFW'98*, pages 27–32. IEEE, 1998.

[Aba99] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[ABFS01] G. Anastasi, A. Bartoli, N. De Francesco, and A. Santone. Efficient verification of a multicast protocol for mobile computing. *The Computer Journal*, 44(1), 2001.

[ABG04] A. Aldini, M. Bravetti, and R. Gorrieri. A process-algebraic approach for the analysis of probabilistic non-interference. *Journal of Computer Security*, 12(2), 2004.

[AFG98] M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. In *Proc. LICS'98*, pages 105–116. IEEE, 1998.

[AG97] M. Abadi and A.D. Gordon. Reasoning about cryptographic protocols in the spi-calculus. In *Proc. CONCUR'97*, volume LNCS 1243, pages 59–73. Springer, 1997.

[And95] H.R. Andersen. Partial model checking. In *Proc. of 10th Symposium Logic in Computer Science*, pages 398–407. IEEE, 1995.

[Arc02] M. Archer. Proving correctness of the basic TESLA multicast stream authentication protocol with TAME. In *Proc. WITS'02*, 2002. Informal proceedings.

[ASW01] W. Arbaugh, N. Shankar, and Y.C. Justin Wan. Your 802.11 wireless network has no clothes. In *Proc. Wireless LANs and Home Networks*. World Scientific, 2001.

[AT91] M. Abadi and M.R. Tuttle. A semantics for a logic of authentication. In *Proc. SPDC'91*, pages 201–216. ACM, 1991.

[Bar98] A. Bartoli. Group-based multicast and dynamic membership in wireless networks with incomplete spatial coverage. *ACM/Baltzer Mobile Networks and Applications*, 3(2):175–188, 1998.

[BB02] S. Buchegger and J. Le Boudec. Performance analysis of the confidant protocol. Cooperation of nodes - fairness in dynamic ad-hoc networks. In *Proc. MobiHoc'02*, pages 226–236. ACM, 2002.

[BDNN01] C. Bodei, P. Degano, F. Nielson, and H.R. Nielson. Static analysis for the pi-calculus with applications to security. *Information and Computation*, 168(1):68–92, 2001.

[Bee03] M.H. ter Beek. Team automata—a formal approach to the modeling of collaboration between system components. Ph.D. thesis, Leiden Institute of Advanced Computer Science, Leiden University, 2003.

[BEKR01] M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg. Team automata for spatial access control. In *Proc. ECSCW'01*, pages 59–77. Kluwer, 2001.

[BEKR03] M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg. Synchronizations in team automata for groupware systems. *Computer Supported Cooperative Work—The Journal of Collaborative Computing*, 12(1):21–69, 2003.

[BG02] M. Boreale and D. Gorla. On compositional reasoning in the spi-calculus. In *Proc. FOSSACS'02*, volume LNCS 2303. Springer, 2002.

[BGW01] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: the insecurity of 802.11. In *Proc. MobiCom'01*, pages 180–189. ACM, 2001.

[BH02] L. Buttyan and J. P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM/Kluwer Mobile Networks and Applications (MONET)*, 8(5), 2002.

[BK03] M.H. ter Beek and J. Kleijn. Team automata satisfying compositionality. In *Proc. FME'03*, volume LNCS 2805, pages 381–400. Springer, 2003.

[BL02] P. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In *Proc. ESORICS'02*, volume LNCS 2502, pages 146–161. Springer, 2002.

[BLP03] M.H. ter Beek, G. Lenzini, and M. Petrocchi. Team automata for security analysis of multicast/broadcast communication. In *Proc. WISP'03*, 2003. Informal proceedings.

[BLP04a] M.H. ter Beek, G. Lenzini, and M. Petrocchi. Team automata for security – a survey. In *Preproceedings SECCO'04*, pages 1–14. Elsevier, 2004. ENTCS – to appear.

[BLP04b] M.H. ter Beek, G. Lenzini, and M. Petrocchi. Team automata for security analysis. Technical Report Tech. Rep. TR-CTIT-04-13, Twente Univ., 2004. DIMACS Workshop Security Analysis of Protocols'04.

[BM89] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Proc. CRYPTO'89*, volume LNCS 435, pages 547–557. Springer, 1989.

[BP76] B. Bell and L. La Padula. Secure computer systems—unified exposition and multics interpretation. Technical Report Tech. Rep. ESD-TR-75-306, MITRE MTR-2997, 1976.

[BPS01] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science, 2001.

[BR04] J. Baeten and M. Reniers. Timed process algebra. In *Proc. SFM'04*, volume LNCS 3185, pages 59–97. Springer, 2004.

[BSSW02] D. Balfanz, D. K. Smetters, P. Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proc. NDSS'02*. The Internet Society, 2002.

[CC02] J. Carmona and J. Cortadella. Input/Output compatibility of reactive systems. In *Proc. FMCAD'02*, volume LNCS 2517, pages 360–377. Springer, 2002.

[CCKM00] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *Proc. ICALP'00*, LNCS 1853, pages 512–523. Springer, 2000.

[CG01] T. Cichocki and J. Gorski. Formal support for fault modeling and analysis. In *Proc. SAFECOMP'01*, volume LNCS 2187, pages 190–199. Springer, 2001.

[CJM00] E.M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, 2000.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–656, 1976.

[DH94] D. Drusinsky and D. Harel. On the power of bounded concurrency I: Finite automata. *Journal of the ACM*, 41(3):517–539, 1994.

[DP03] N. De Francesco and M. Petrocchi. Authenticity in a reliable protocol for mobile computing. In *Proc. SAC'03*, pages 318–324. ACM, 2003.

[Dub86] C. Duboc. Mixed product and asynchronous automata. *Theoretical Computer Science*, 42:183–199, 1986.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. Inf. Theory*, 29(2):198–208, 1983.

[Ell97] C.A. Ellis. Team automata for groupware systems. In *Proc. GROUP'97*, pages 415–424. ACM, 1997.

[EP04] L. Egidi and M. Petrocchi. Modelling a secure agent with team automata. In *Preproceedings VODCA'04*, pages 119–134. Elsevier, 2004. ENTCS – to appear.

[ES00] N. Evans and S. Schneider. Analysing time-dependent security properties in CSP using PVS. In *Proc. ESORICS'00*, volume LNCS 1895, pages 222–237. Springer, 2000.

[FG94] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1994.

[FG95] R. Focardi and R. Gorrieri. A taxonomy of security properties for process algebras. *Journal of Computer Security*, 3(1):5–34, 1995.

[FG97a] R. Focardi and R. Gorrieri. The compositional security checker—a tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.

[FG97b] R. Focardi and R. Gorrieri. The compositional security checker. A tool for the verification of information flow security Properties. *IEEE TSE*, 27(3):550–571, 1997.

[FG01] R. Focardi and R. Gorrieri. Classification of security properties—part II: Information flow. In *Proc. FOSAD 2001—Tutorial Lectures*, volume LNCS 2171, pages 331–396. Springer, 2001.

[FGM00a] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proc. ICALP'00*, volume LNCS 1853, pages 354–372. Springer, 2000.

[FGM00b] R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as non interference. In *ENTCS 32*, 2000.

[FGM03] R. Focardi, R. Gorrieri, and F. Martinelli. Real-time information flow analysis. *Journal of Selected Areas of Communications*, 21(1):20–35, 2003.

[FGM04] R. Focardi, R. Gorrieri, and F. Martinelli. Classification of security properties—part I: Network security. In *Proc. FOSAD 2001/2002—Tutorial Lectures*, volume LNCS 2946, pages 139–185. Springer, 2004.

[FJTG99] J. C. Herzog F. J. Thayer and J. D. Guttman. Strand spaces: proving security protocols correct. *Journal of Computer Security*, 7(1):191–230, 1999.

[FKK96] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL protocol version 3.0 - Internet Draft, 1996. http://wp.netscape.com/eng/ssl3/ssl-toc.html.

[FM99] R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proc. FM'99*, volume LNCS 1708, pages 794–813. Springer, 1999.

[FMS01] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Proc. Workshop on Selected Areas in Cryptography*, volume LNCS 2259, pages 1–24, 2001.

[GJ01] A.D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proc. CSFW'01*, pages 145–159. IEEE, 2001.

[GJ02] A.D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. In *Proc. CSFW'02*, pages 77–91. IEEE, 2002.

[GLM03] R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for real-time cryptographic protocol analysis. In *Proc. ESOP'03*, volume LNCS 2618, pages 114–128. Springer, 2003.

[GM82] J.A. Goguen and J. Meseguere. Security policy and security models. In *Proc. S&P'82*, pages 11–20. IEEE, 1982.

[GM01] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *Proc. NDSS'01*. The Internet Society, 2001.

[GM03] R. Gorrieri and F. Martinelli. Process-algebraic frameworks for the specification and analysis of cryptographic protocols. In *Proc. MFCS'03*, volume LNCS 2747, pages 46–67. Springer, 2003.

[GMP04] R. Gorrieri, F. Martinelli, and M. Petrocchi. A formalization of credit and responsibility within the GNDC schema. In *Proc. SASYFT'04*, 2004. Informal proceedings.

[GMPV01] A. Giani, F. Martinelli, M. Petrocchi, and A. Vaccarelli. A case study with PaMoChSA: a tool for the automatic analysis of cryptographic protocols. In *Proc. SCI-ISAS'01*, volume 5. IIIS, 2001.

[GMPV03a] R. Gorrieri, F. Martinelli, M. Petrocchi, and A. Vaccarelli. Compositional verification of integrity for digital stream signature protocols. In *Proc. ACSD'03*, pages 142–149. IEEE, 2003.

[GMPV03b] R. Gorrieri, F. Martinelli, M. Petrocchi, and A. Vaccarelli. Formal analysis of some timed security properties in wireless protocols. In *Proc. FMOODS'03*, volume LNCS 2884, pages 139–154. Springer, 2003.

[GOR02] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Role based specification and security analysis of cryptographic protocols using asynchronous product automata. In *Proc. DEXA'02*, pages 473–482. IEEE, 2002.

[GR01] R. Gennaro and P. Rohatgi. How to sign digital streams. *Information and Computation*, 165(1):100–116, 2001.

[GT00] J. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. CSFW'00*, pages 24–34. IEEE, 2000.

[Har87] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459: Internet X.509 public key infrastructure certificate and CRL profile, IETF, 1999. http://www.ietf.org/rfc/rfc2459.txt.

[HH94] T. Hirst and D. Harel. On the power of bounded concurrency II: Pushdown automata. *Journal of the ACM*, 41(3):540–554, 1994.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Hol03] G.J. Holzmann. *The SPIN Model Checker—Primer and Reference Manual*. Addison Wesley, 2003.

[HPJ02] Y. Hu, A. Perrig, and D.B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proc. Mobicom 2002*, pages 12–23. ACM, 2002.

[HR95] M. Hennessy and T. Regan. A temporal process algebra. *Information and Computation*, 117:222–239, 1995.

[JMB01] D. Johnson, D. Maltz, and J. Broch. *DSR - the dynamic source routing protocol for multihop wireless ad hoc networks*. Addison-Wesley, 2001.

[Kle03] J. Kleijn. Team Automata for CSCW – A survey –. In *Petri Net Technology for Communication-Based Systems—Advances in Petri Nets*, volume LNCS 2472, pages 295–320. Springer, 2003.

[KW96] D. Kindred and J.M. Wing. Fast automatic checking of security protocols. In *Proc. 2nd Usenix Workshop on Electronic Commerce*, pages 41–52, 1996.

[LA01] C. Lopes and P. Aguiar. Aerial acoustic communications. In *Proc. WASPAA'01*. IEEE, 2001.

[Lam79] L. Lamport. Constructing digital signatures from a one-way function. Technical Report CSL 98, SRI Intl, 1979.

[LEF+00] M. Lamming, M. Eldridge, M. Flynn, C. Jones, and D. Pendlebury. Providing access to any document, any time, anywhere. *ACM Transactions on Computer-Human Interaction*, 7(3):322–352, 2000.

[LEH03] Y.W. Law, S. Etalle, and P. Hartel. Assessing security in energy-efficient sensor networks. In *Proc. SEC'03*, pages 459–463. Kluwer, 2003.

[LGL03] G. Lenzini, S. Gnesi, and D. Latella. Spider: a Security Model Checker. In *Proc. FAST'03*, pages 163–180. Also, Technical Report ITT-CNR-10, 2003. Informal proceedings.

[LMMN05] X. Liu, C. Madson, D. McGrew, and A. Nourse. Internet Draft: draft-nourse-scep-11, Cisco Systems, 2005. http://www.ietf.org/internet-drafts/draft-nourse-scep-11.txt.

[LMSP00] R. Lanotte, A. Maggiolo-Schettini, and A. Peron. Timed cooperating automata. *Fundamenta Informaticae*, 42:1–21, 2000.

[LMST03] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Weak bisimulation for probabilistic timed automata and applications to security. In *Proc. SEFM'03*, pages 34–43. IEEE, 2003.

[Low95] G. Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, 1995.

[Low96] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In *Proc. TACAS'96*, volume LNCS 1055, pages 147–166. Springer, 1996.

[Low98] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proc. CSFW'98*, pages 96–105. IEEE, 1998.

[LPW03] B. Lamparter, M. Plaggemeier, and D. Westhoff. About the impact of co-operation approaches for ad hoc networks. In *Proc. MobiHoc'03*. ACM, 2003.

[LR97] G. Lowe and A.W. Roscoe. Using CSP to detect errors in the TMN protocol. *Software Engineering*, 23(10):659–669, 1997.

[LT89] N. Lynch and M.R. Tuttle. An introduction to Input/Output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[Lyn99] N. Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proc. CSFW'99*, pages 14–31. IEEE, 1999.

[Mar03] F. Martinelli. Analysis of security protocols as open systems. *Theoretical Computer Science*, 290(1):1057–1106, 2003.

[Mea95] C. Meadows. Formal verification of cryptographic protocols: a survey. In *Proc. ASIACRYPT'94 – Advances in Cryptology*, volume LNCS 917, pages 135–150. Springer, 1995.

[MGLB00] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehaviour in mobile ad hoc networks. In *Proc. MobiCom'00*, pages 255–265. ACM, 2000.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. LNCS 92. Springer, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[MM99] D. Marchignoli and F. Martinelli. Automatic verification of cryptographic protocols through compositional analysis techniques. In *Proc. TACAS'99*, volume LNCS 1579, pages 148–162. Springer, 1999.

[MM02a] P. Michardi and R. Molva. Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Proc. CMS'02*, pages 107–121. Kluwer, 2002.

[MM02b] P. Michiardi and R. Molva. Simulation-based analysis of security exposures in mobile ad hoc networks. In *Proc. European Wireless'02*, 2002.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proc. S&P'97*, pages 141–153. IEEE, 1997.

[MPV02] F. Martinelli, M. Petrocchi, and A. Vaccarelli. Automated analysis of some security mechanisms of SCEP. In *Proc. ISC'02*, volume LNCS 2433, pages 414–427. Springer, 2002.

[MPV03] F. Martinelli, M. Petrocchi, and A. Vaccarelli. Compositional verification of secure streamed data: a case study with EMSS. In *Proc. ICTCS'03*, LNCS 2841, pages 383–396. Springer, 2003. Also, informally presented at WITS'03.

[MPV04] F. Martinelli, M. Petrocchi, and A. Vaccarelli. Local management of credits and debits in mobile ad hoc networks. In *Proc. CMS'04 – to appear*. Kluwer, 2004. Also, informally presented at UK-Ubinet'04.

[NNHJ99] F. Nielson, H. Nielson, R. Hansen, and J. Jensen. Validating firewalls in mobile ambients. In *Proc. CONCUR'99*, volume LNCS 1664, pages 463–477. Springer, 1999.

[NPW02] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL—A proof assistant for higher-order logic*. LNCS 2283. Springer, 2002.

[NS78] R. Needham and M. Schroeder. Using encryption for authentification in large networks of computers. *Communications of the ACM*, 21(12), 1978.

[Ohe03] D. von Oheimb. Interacting State Machines – a stateful approach to proving security –. In *Proc. FASec'02*, LNCS 2629, pages 15–32. Springer, 2003.

[OL02] D. von Oheimb and V. Lotz. Formal security analysis with interacting state machines. In *Proc. ESORICS'02*, LNCS 2502, pages 212–228. Springer, 2002.

[otICS99] L.M.S.C. of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. In *IEEE Standard 802.11, 1999 Edition*, 1999.

[Pau97] L.C. Paulson. Proving properties of security protocols by induction. In *Proc. CSFW'97*, pages 70–83. IEEE, 1997.

[PCS02] J. M. Park, E. K. P. Chong, and H. J. Siegel. Efficient multicast packet authentication using signature amortization. In *Proc. S&P'02*, pages 227–240. IEEE, 2002.

[PCST01] A. Perrig, R. Canetti, D. X. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Proc. NDSS'01*. The Internet Society, 2001.

[PCTS00] A. Perrig, R. Canetti, D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. S&P'00*, pages 56–73. IEEE, 2000.

[Pos80] J. Postel. The User Datagram Protocol - RFC 768, 1980.

[PST+02] A. Perrig, R. Szewczyk, D. Tygar, V. Wen, and D. E. Culler. SPINS: security protocols for sensor networks. *Wireless Networks Journal*, 8:521–534, 2002.

[RG97] A.W. Roscoe and M.H. Goldsmith. The perfect spy for model-checking crypto-protocols. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[Rog91] P. Rogaway. *The round complexity of secure protocols*. PhD thesis, MIT, Cambridge, Massachusetts, 1991.

[Rom01] K. Romer. Time synchronization in ad hoc networks. In *Proc. MobiHoc'01*, pages 173–182. ACM, 2001.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Comm. of the ACM*, 21(2):120–126, 1978.

[RSG+00] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The modelling and analysis of security protocols: the CSP approach*. Addison-Wesley, 2000.

[SA99] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proc. Security Protocols Workshop*, volume LNCS 1796, pages 172–194. Springer, 1999.

[SBHJ03] N.B. Salem, L. Buttyan, J.P. Hubaux, and M. Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks. In *Proc. MobiHoc'03*, pages 13–24. ACM, 2003.

[Sch96] B. Schneier. *Applied Cryptography*. J. Wiley & sons, Inc, 1996.

[Sch00] F.B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[SIR02] A. Stubblefield, J. Ioannidis, and A. D. Rubin. Using the Fluhrer, Mantin and Shamir attack to break WEP. In *Proc. NDSS'02*. The Internet Society, 2002.

[SOR93] N. Shankar, S. Owre, and J.M. Rushby. *PVS Tutorial*, 1993. Tutorial Notes, FME'93: Industrial-Strength Formal Methods.

[SS98] V. Shmatikov and U. Stern. Efficient finite state analysis for large security protocols. In *Proc. CSFW'98*, pages 105–116. IEEE, 1998.

[ST98] T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. Mobile Agents and Security*, LNCS 1419, pages 44–60. Springer, 1998.

[Sto98] S. Stoller. A reduction for automated verification of authentication protocols. Technical Report Tech. Rep. 520, Computer Science Dept., Indiana Univ., 1998.

[tBB] M.H. ter Beek and R.P. Bloem. Model checking team automata for access control. Unpublished manuscript, 2003.

[UBG03] A. Urpi, M. Bonuccelli, and S. Giordano. Modelling cooperation in mobile ad-hoc networks: a formal description of selfishness. In *Proc. WiOpt'03*, 2003. Informal proceedings.

[Yao86] A.C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986.

[ZCY03] S. Zhong, J. Chen, and Y. Yang. Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proc. Infocom'03*. IEEE, 2003.

[ZH99] L. Zhou and Z.J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.

Appendix A

A secure protocol for mobile computing

A.1 Introduction

This appendix is devoted to the description of a general architecture for giving authenticity to a reliable protocol for mobile computing.

Technological developments in computer and communication are enabling the deployment of computing systems based on portable computers and wireless networking. Users may be equipped with hand-held computing devices and roam around freely while maintaining connectivity with a wired infrastructure. Such architectures may be exploited for novel applications and services spread out in a variety of directions.

With the considerable spread of wireless access both undisputed advantages and new security problems arise. Indeed, broadcasting messages over radio channels makes traffic eavesdropping and packets' injection relatively easy for adversaries in possession of adequate resources. The practicality of security attacks in wireless environments has been discussed and shown recently, [BGW01, SIR02].

We consider a simplified version of the multicast protocol for mobile computing developed in [AB99, Bar98]. Design issues are considered in the cited papers in order to support reliable and totally ordered communication within a set of processes, running on mobile hosts. The protocol is concerned with *r*eliable multicast communication, where *reliability* means, very informally, that all packets (messages) are delivered and that duplicates are discarded. Each process at stake is able to detect possible losses of packets. Lost packets are then recovered

by using a mechanism based on *nack* messages and retransmission. The protocol is not intended to support real time applications.

The design of the protocol does not take in any account security issues and the presence of possible adversaries has not been considered. In this paper we assume that a set of *legitimate hosts* are authorized to participate through the protocol. Unauthorized mobile hosts will be treated as potential adversaries. We propose an extension of the original protocol with authentication mechanisms to assure the capability for the legitimate hosts to authenticate a packet as originated by a legitimate host. The proposed protocol could be fruitfully used in environments where authenticity and integrity are the main concerns, whereas secrecy is less important. For example, taking into consideration a university environment, a professor and his assistants, whilst exchanging students grades, are more worried about the integrity rather than the secrecy of the information exchanged. We base the extension on some concepts coming from proposals by Balfanz *e*t al., [BSSW02], which seem to fit our context quite well. Further, an analogy between digital streams considered in [GR01] and the messages exchanged through the protocol will be highlighted.

In the following, an overview of the original protocol is given. Then, upon considering the protocol in a security environment, a new version will be presented, enriched with security mechanisms fulfilling authenticity.

A.2 Protocol overview

A very intuitive notation will be used throughout the appendix. The sending and reception of a message msg from a single sender A to a single receiver B is represented by:

$$c_j \quad A \xrightarrow{u} B \quad : \quad msg$$

where c_j is the j-th communication channel. Raised u stands for "unicast" modality (a point to point connection).

We represent the multicast of a message (from a sender A to a set $\mathcal{B}$ of multiple but defined receivers, with only one sending action) as follows:

$$c_j \quad A \xrightarrow{m} \mathcal{B} \quad : \quad msg$$

where $\mathcal{B}$ is a set of processes. Raised m stands for "multicast" modality.

We represent the broadcast of a message (from a sender A to multiple receivers $\mathcal{B}$, with only one sending action) as follows:

$$c_j \quad A \xrightarrow{b} \mathcal{B} \quad : \quad msg$$

where A possibly broadcasts msg to each process B belonging to the set $\mathcal{B}$. Contrary to the multicast modality, here the set $\mathcal{B}$ may change dynamically and it is not fixed in advance. Prime b stands for "broadcast" modality.

For a sketch of the original protocol, we refer to the simplified version in [ABFS01]. The full version can be found in [Bar98], along with a detailed discussion of its motivation and advantages.

The system model on which the protocol is defined is as follows. It consists of mobile hosts MHs and stationary hosts SHs, called *gateways*. The gateways are connected both to a wired network (that provides reliable and FIFO-ordered communication) and to a wireless link that covers a spatially limited *cell* nearby each gateway. Cells provide only incomplete coverage and wireless communication is unreliable. MHs communicate through wireless links and may move. Movements are unpredictable, in the way that a MH may leave a cell without prior negotiation and reenter any other cell or even remain out of coverage for some time. The protocol establishes that MHs may only exchange messages with the gateway of the cell where they happen to be located in and with a special stationary host acting as the coordinator. The gateways may broadcast messages to all MHs in their cell and send messages to a specific MH in their cell. The resulting scenario is quite general since it can accommodate contemporary wireless LANs, infrared networks, picocellular wireless networks where cells coincide with rooms in a building, physical obstructions and long-range movements.

The protocol works as follows. A dedicated SH acts as the *c*oordinator, denoted as C. A mobile host may receive messages from the application layer and send them to the other hosts. Such messages are sent by the mobile host as *n*ew messages to the coordinator C that processes incoming *n*ew messages in sequence. C constructs a message containing the payload and an increasing sequence number. C then transmits the resulting message to all gateways through a FIFO-multicast. Gateways broadcast this message in their respective cells.[1]

Due to their movement across cells and uncovered areas and to the unreliability of the wireless links, MHs could receive duplicates or could miss packets.

The exchange of a *new* message can be formalized as in Fig. A.1 and the procedure can be explained as follows:

1. A mobile host MH, wishing to communicate a *new* message to others, sends the message to the coordinator C.

[1] Actually, the full version of the protocol is based on a set of coordinators whereas here a global synchronization structure among the coordinators has been considered.

$$
\begin{array}{lll}
c_1 & MH \xrightarrow{u} C & : \; new \\
c_2 & C \xrightarrow{m} \{G_1, G_2, ..., G_N\} & : \; new, seq \\
c_3 & G_i \xrightarrow{b} \{MH \mid MH \, is \, in \, cell \, i\} & : \; new, seq
\end{array}
$$

Figure A.1: Transmitting a *new* message.

2. The coordinator multicasts *new* to only all the static hosts $\{G_1, .., G_N\}$ on the wired link. It adds to the message the tag seq, containing the sequence number of the current *new*. Each gateway G_i maintains a list of messages recently received from C.
3. Each gateway G_i, responsible for cell i, broadcasts what it previously received from C in the cell and the mobile hosts currently present in cell i receive the message.

By maintaining a history of the received sequence numbers, a mobile host discards duplicates and sends the gateway a proper *nack* message upon receiving an out-of-order message. Upon receiving a *nack*, the gateway sends MH a copy of the missing multicasts. Each gateway stores a copy of each multicast previously sent until it knows that the multicast has been delivered to every mobile host.

The protocol does not use any notion of hand-off, *i.e.*, it does not require any data exchange between the old gateway and the new one when a host moves from one cell to another.

A.3 The secured protocol

A.3.1 Hostile environment

Hereafter, we consider a set of *legitimate hosts*, denoted as LMHs, authorized to participate through the protocol plus an unauthorized mobile adversary with the adequate technical equipment to eavesdrop on traffic and actively inject packets over the wireless links. For details about the practicality of such interferences we refer to [BGW01, SIR02].

The protocol was originally designed to guarantee a set of properties, among which: i) *Integrity*, requiring that any packet received by a legitimate host has

been originated by a legitimate host; ii) *No Duplicate*, stating that no legitimate host accepts duplicate packets, i.e. duplicates are discarded.

The properties of the protocol have been formally analyzed in [ABFS01]. However, these properties might not hold in a classical context of security analysis where the presence of an adversary has to be considered.

We now define two properties regarding authentication between legitimate hosts and stationary hosts:

(P_A) Capability for the coordinator to authenticate the sender of a *new* message as a legitimate host.

(P_B) Capability for all the legitimate hosts to authenticate the received broadcasts as indeed originated from the stationary host responsible for the cell in which they happen to be located.

We do not require that a gateway correctly identifies a LMH when it asks for a lost packet sending a *nack* message. The authentication of origin of the *nack* message is unimportant given that the contents of the packets do not have to be kept secret.

We propose to add security procedures to the original protocol to make properties P_A and P_B hold.

With reference to asymmetric cryptography [Sch96], the digital signature is the typical mechanism to guarantee authentication of origin and integrity. In our context, unfortunately, to digitally sign each *new* message may cause an infeasible computational overload for mobile hosts which have intrinsic limited resources. Hence, we look for solutions with thrifty use of classical digital signature schemes as in [RSA78].

Since we use public key cryptography, we need a method for guaranteeing the ownership of the public keys at stake. Common solutions rely on Public Key Infrastructures (PKIs) and digital certificates, [HFPS99]. We prefer not to use digital certificates, since they may result as a bottleneck for the whole system. Subsection A.3.2 presents an alternative method for bootstrapping authentication without the need of a PKI.

A.3.2 Bootstrapping authentication

We make use of a method for bootstrapping authenticated and integral communication between mobile hosts participating in the protocol. A *pre-authentication*

phase, in which a certain amount of information is exchanged between legitimate hosts and SHs over a privileged channel will be inserted. Information exchanged during the *pre-authentication* phase will be used through the main wireless link.

We inherit the concept of *Location-Limited Channels* from [BSSW02]. A *Location-Limited Channel* (hereafter LLC) is separated from the main wireless link and exploits security properties by virtue of the media over which data are sent. In order to be used for *pre-authentication*, LLCs must support *physical identification*, *i.e.*, human operators must be able to visibly control which devices are communicating with each other during a transmission over the LLC. Hence audio and infrared channels could be good LLCs given the physical limited range of their transmissions, [LEF+00, LA01].

We use LLCs to exchange information about the public keys of the stationary hosts. The *physical identification* property of the transmission over LLCs is a smart loophole to bypass the need of a PKI to guarantee the authenticity of the public keys at stake (to know more about the need for a right association between a public key and its owner, the interested reader can see Chapter 5 of this thesis). The physical proximity of the hosts during the transmission over the LLC (and the consequent monitoring) is a way out to delegate the hosts themselves as guarantors for the benign nature of data exchanged over the LLC. Subsequent communications over the main wireless link will be accepted as "well-originated" if they refer to the data exchanged over the LLC. The practicability of such pre-authentication schemes in wireless environments has been successfully shown in [BSSW02].

A.3.3 Assumptions

To distinguish between authorized and unauthorized hosts, we write LMH to denote a legitimate host. Each LMH is also a mobile host MH but the opposite is not true: a generic MH is not necessarily a legitimate host.

We assume that the multicast over channel c_2 in Fig. A.1 can not be compromised. We trust the agents on the wired link. Furthermore, we do not consider an adversary able to tamper with the communication on the wired link. If there is an injection of data coming from unauthorized hosts, we assume it to occur on the wireless link.

We use LLCs with the following assumptions: i) of all the mobile hosts, *only* the legitimate hosts can transmit over the LLC; ii) the stationary hosts can transmit over the LLC and hold a pair of public/private keys to perform regular signature schemes as in [RSA78].

The environment under examination consists of both wired and wireless links.

Communications necessarily pass through the stationary hosts on the wired link. This architecture allows us to separate the authentication mechanism into two distinguished parts: the first part concerns authenticating a legitimate host to the coordinator, while the second is concerned with the authentication of the gateways G_i to the legitimate hosts currently present in cell i. Even though we loose the precise identity of the sender of a new message, we are interested only in *g*eneric authenticity (each legitimate host can recognize whether a message was sent by a legitimate host) rather than *s*ource authenticity (the capability to identify the single party within a set).

A.3.4 Authenticating the mobile sender

We require a mobile host to prove its legitimacy in order to send a message. In the pre-authentication phase the host has to be physically located close to the coordinator.

$$\begin{array}{llcl} LLC & LMH \xrightarrow{u} C & : & Hash\{Nonce_{LMH}\} \\ LLC & C \xrightarrow{u} LMH & : & pk_C \end{array}$$

The pre-authentication phase takes place over a selected LLC (*e.g.*, exploiting infrared technology). First, the mobile host transmits the digest of a randomly generated number $Nonce_{LMH}$ to the coordinator over the LLC. The coordinator replies transmitting its public key pk_C. An adversary able to listen over the LLC does not add any useful information to his knowledge, given the public nature of the information exchanged from C to LMH. (The non-reversibility of one-way hash functions is implicitly assumed too.)

$$\begin{array}{llcl} c_0 & LMH \xrightarrow{u} C & : & \{Nonce_{LMH}\}_{pk_C} \\ c_1 & LMH \xrightarrow{u} C & : & Hash\{new, Nonce_{LMH}, Ndup\}, \\ & & & new, Ndup \end{array}$$

Communication continues over the main wireless link. Communication over channel c_0 has been added compared with the original protocol: LMH sends the encryption of $Nonce_{LMH}$ with C's public key $\{Nonce_{LMH}\}_{pk_C}$. The contents of the message over channel c_1 in Fig. A.1 have been changed by applying a one-way hash function to the 3-tuple consisting of the payload *new*, the nonce and another nonce $Ndup$ to be used only once.

How can the coordinator have guarantees about the origin of the message? C can decrypt message over c_0 with its private key and retrieve $Nonce_{LMH}$. Then,

it computes the digest of the nonce and compares it with that received over the LLC. If the two digests match, C may be reasonably sure that whoever sent $\{Nonce_{LMH}\}_{pk_C}$ over channel c_0 is the same mobile host that previously transmitted over the LLC. Further, the whole message is authenticated as coming from the same mobile host, since $Nonce_{LMH}$ is introduced as an argument of a one-way function together with *new* (and $Ndup$). In this way, *new* is tied to $Nonce_{LMH}$. From assumption i) on LLCs (see this section), it follows that the mobile host that originated the message over c_1 is indeed a legitimate host.

The nonce $Ndup$ is inserted to avoid replay attacks: unauthorized hosts could eavesdrop on channel c_1 and simply transmit the same message later. Each time LMH sends a *new* message, he should randomly generate a nonce $Ndup$ to insert in the packet both as plaintext and as an argument of the hash function. C should record the $Ndup$ he receives and should not accept any message with the same $Ndup$ in the future.

We get into the issue of defining the expiry period for the information sent from LMH to C over the LLC. We suggest that LMH generates an explicit request to invalidate the nonce $Nonce_{LMH}$. This request may be sent over the LLC and may contain the nonce as a clear text seeing that once the nonce has been invalidated its knowledge on behalf of an adversary is irrelevant. On the other hand, the very first request reasonably comes from the legitimate LMH, the one (except the coordinator) to know $Nonce_{LMH}$.

In the construction above, LMH performs a public key encryption only once. Further, there is no connection between this construction and the movement of hosts from one cell to another: the transmission over LLC and channel c_0 happens once only before the first packet is transmitted. There is no relation to the gateways of a single cell.

A.3.5 The broadcast environment

Contrary to above, what will be proposed now is a sort of *authentication on demand*. Each mobile host maintains its capability to receive broadcasted messages apart from the fact that the gateways authenticate themselves to it. It is reasonable to suppose that a LMH decides to trust a packet as sent by a legitimate gateway or to willingly ask for an authenticity proof.

In the latter case we suggest to exploit part of a mechanism originally developed to sign digital streams, [GR01]. In Chapter 3 of this thesis we have discussed the off-line case of that mechanism. Here, we focus and briefly describe the online case, that is suitable when the sender of the stream does not know the content

of the stream in advance (*e.g.*, , a live broadcast).

Similarities between digital streams and the finite packets exchanged through our protocol are straightforward to highlight: i) with regard to authentication techniques they both require little use of traditional signature schemes (the stream receiver has to check the signature as the packets arrive, the mobile hosts may not have the resources to perform public key operations in their completeness); ii) since each gateway is devoted to simply forwarding packets coming from the coordinator, it does not know the contents of the packets in advance, as in live broadcast.

Each forwarded packet will be treated as a block belonging to a digital stream, see [GR01].

In the pre-authentication phase, LLC is used to transmit a first "1-time" public key from the gateway to the petitioning LMH. 1-time signature schemes are a special kind of signature scheme introduced in [Lam79], much faster to compute and verify than regular signatures. These schemes can be used to sign only one packet. We assume that each gateway can generate an arbitrary number of 1-time public keys.

$$\begin{array}{lll} LLC & LMH \xrightarrow{u} G_i & : \quad request \\ LLC & G_i \xrightarrow{u} LMH & : \quad 1pk_{G_i}^{seq} \end{array}$$

A legitimate host that wants authenticated packets asks for the transmission of the first 1-time public key of the gateway responsible for the cell in which the host happens to be located. This transmission happens over the LLC. (For the transmission over the LLC the host is assumed to be close to the gateway.) With notation $1pk_{G_i}^{seq}$ we indicate the *s*eq-th 1-time public key of G_i, where seq is the sequence number of the packet the gateway is to broadcast in the cell (the same seq as in the original protocol, Fig. A.1).

$$\begin{array}{ll} c_3 & G_i \xrightarrow{b} \{MH | MH\, is\, in\, cell\, i\} : new, seq, 1pk_{G_i}^{seq+1}, \\ & Sig^{-1}_{1pk_{G_i}^{seq}}\{Hash\{new, seq, 1pk_{G_i}^{seq+1}\}\} \end{array}$$

G_i broadcasts in its cell the (new, seq) as in the original protocol in Fig. A.1 along with a 1-time signature of its hash based on the 1-time public key sent over the LLC. (With notation $Sig^{-1}_{1pk^j_{G_i}}\ \{msg\}$ we mean: "*m*sg is signed with the private key corresponding to the j-th 1-time public key of G_i.") A new 1-time public key $1pk_{G_i}^{seq+1}$ is also transmitted and will be used to verify the signature of the $seq + 1$ broadcasted message. This structure is repeated for all the packets gateway G_i broadcasts in its cell.

Contrary to what proposed in Subsection A.3.4, there is no need for the insertion of a nonce to prevent replay attacks. seq plays the role of the nonce $Ndup$ in the previous construction. seq can not be manipulated since it is an argument of the 1-time signature.

Broadcast communication is received by everybody in the cell but the verification of the 1-time signature is likely to be taken into consideration only by the hosts who have previously requested the first 1-time public key over the LLC. The other LMHs do not take into account the signature and simply consider the payload new and the sequence number seq.

To work correctly, the whole mechanism requires that no packet is lost. The sending of "nack messages" already considered by the protocol under investigation guarantees such a requirement. Suppose a host receives a packet containing a sequence number greater than expected: according to the original protocol, LMH asks for the re-transmission of the lost packets (see Section A.2). LMH can rebuild the correct order for verifying the signature because each 1-time public key is strictly related to the sequence number of the packets: the seq-th packet contains the (seq + 1)-th public key, to be used to verify the signature of the (seq + 1)-th packet, and so on.

We give an informal justification on the correctness of the secured construction. Suppose a generic LMH, at the n-th point of the computation, accepts the following as an authentic message:

$$new^X, n, 1pk_{G_i}^{n+1},$$
$$Sig_{1pk_{G_i}^n}^{-1}\{Hash\{new^X, n, 1pk_{G_i}^{n+1}\}\}$$

where new^X is originating from an unauthorized mobile host. If the generic LMH accepts this message as being authentic, it means that i) the adversary knows the private key corresponding to the n-th 1-time public key of the gateway and consequently he is able to reproduce valid signatures (which shouldn't be possible seeing that private information of the stationary hosts are never exchanged during the protocol) or ii) the adversary was able to forge the authentication chain by inserting in message (n-1)-th his own 1-time public key. This is possible only if the unauthorized host is able to insert his own 1-time public key at the very first communication over LLC. Taking into consideration the assumption on transmission over LLCs, it follows that this possibility, in reality, is infeasible.

The adversary could be able to inject over cell i a broadcast coming from a stationary host responsible for another cell. A generic LMH in cell i does not accept this message as an authentic message since he does not have the right 1-time public key to verify the signature (the last authentication chain in which he

has involved is the chain related to the stationary host in which he happens to be located).

A.4 Related work

The Wired Equivalent Protocol (WEP) has been included in the 802.11 standard [otICS99] for wireless LANs as an attempt to solve security problems of wireless connectivity. The primary goal of WEP is to protect the confidentiality of user data from eavesdropping. A related goal concerns with access control, *i.e.*, how to prevent the injection of new traffic from unauthorized mobile hosts. To this aim, the 802.11 standard includes an optional feature to discard all packets not encrypted according to WEP. In reality we are not interested in the secrecy of the exchanged packets, but rather reverting to WEP in order to achieve authenticity of origin. Unfortunately, WEP contains security flaws that give rise to a number of vulnerabilities prone to attacks, [ASW01, BGW01, FMS01, SIR02].

The use of out-of-band channels to bootstrap authentication in wireless networks was first proposed by Anderson and Stayano in [SA99]. Their *Resurrecting Duckling* protocol sets up a relationship between two devices, in their terminology a mother and a duckling. In the initial phase of the protocol the two devices exchange a secret key over a LLC established through *physical contact*. Successively, the duckling uses the secret key to recognize its mother over the wireless link. In [BSSW02] Balfanz *et al.* extend the concept of LLCs not only to set up a master-slave relationship but they consider LLCs to be generally used for ad-hoc wireless networks. To build up authentication mechanisms for the protocol under investigation, we have chosen to avoid the restrictive condition of physical contact for LLCs in order to exploit in the pre-authentication phase the wireless capability of both the stationary and the mobile hosts.

Analogously to [BSSW02], we do not require our LLC to be resistant to eavesdropping. On the contrary, the Resurrecting Duckling protocol of [SA99] expects a shared secret key to be exchanged over the LLC. This makes the LLC vulnerable to eavesdropping. Being the shared key compromised, all subsequent communications on the main wireless link could be compromised, i.e. an adversary could obtain the information necessary to impersonate someone else. The usage of public key cryptography renders the channel cold to passive eavesdropping over the LLC because of the public nature of the information exchanged. Contrary to [BSSW02, SA99], our environment does not properly follow the definition of an *"ad-hoc wireless network"*: transmissions of meaningful payloads do not

take place entirely over wireless links, nor are mobile routers present in the system to forward messages to final mobile receivers. Communications necessarily pass through the stationary hosts on the wired link. Bootstrapping authentication through pre-authentication over LLCs can be applied to more general scenarios than peer to peer authentication in ad-hoc wireless networks.

A.5 Summary

Starting from a known protocol for distributed mobile systems, we have added authentication mechanisms over the wireless links. The mechanisms rely on two different techniques: secure wireless channels to initialize the communications and "1-time" signature schemes. These techniques have been mainly chosen to avoid the need for a Public Key Infrastructure and for the low complexity of the underlying encryption/decryption algorithms.

Appendix B

Selfishness in mobile ad hoc networks

B.1 Introduction

In this appendix, we briefly introduce the notion of selfishness in mobile ad hoc networks. Then, we present a scheme for monitoring possible selfish behaviors.

Unlike traditional mobile networks, ad hoc networks do not rely on any wired infrastructure. Instead, the network is kept connected by the mobile hosts. In order to make a mobile network functional, the nodes need to be self-organized, in such a way that a message is delivered from a source to a destination through a set of intermediate nodes. The deployment of ad hoc networks for civilian applications is taking a footing. In such applications, the nodes are not governed by a single authority and need not share a common goal (the contrary could be the case in emergency and military applications). Thus, cooperative behaviours, such as forwarding each other's packets, cannot be easily assumed. The single nodes could prefer to save battery life for their own communication, rather than to forward packets for other nodes. Such an attitude is denoted in the recent literature as *selfishness* of the node. Simulation results (see, *e.g.*, [MM02b]) have recently pointed out that a selfish behaviour can be as harmful, in terms of the network throughput, as a malicious one.

There is a growing interest in the research community for detecting and preventing a selfish behaviour, and promoting cooperation between nodes, (see, *e.g.*, [BB02, BH02, MGLB00, MM02a, SBHJ03, ZCY03]). Here, we propose an infrastructure for a local management of credits (*i.e.*, a measure of how many pack-

ets node A has forwarded for node B) and debits (*i.e.*, a measure of how many packets node B has forwarded for node A). Each node maintains this information in a local repository that we call *credit table*, on which the node can rely to judge the past behaviour of the other nodes in the network. More specifically, we define rules for the table initialization, its maintenance, and secure acknowledgments testifying the actual forwarding of packets in the network.

We propose to use network-layer acknowledgments (additional data in routing protocols specifications like [JMB01]) to provide to the packet source an authenticated proof that the packet has been delivered to destination. We specify the structure for the acknowledgment request and the corresponding acknowledgment. Then, we introduce a mechanism that amortizes the signaling of "occurred delivery" over blocks of *n* data packets, thus reducing the communication overhead on the way back from destination to source. Further, we deal with some kind of attacks to which our scenario is prone.

B.2 The credit table

A table called the *credit table* (CT) is maintained at each node's side. Rows in the table consist of triples (*h(UC)*, *# debs*, *# creds*), where $h(UC)$ is the hash value of the identifier of the node, and *# creds* and *# debs* are the current values of the credits and debits counter related to that node.

Who maintains the table, say node *A*, quantifies the good behaviour of the node corresponding to *h(UC)*, say node *B*, with respect to *B*'s past attitude to forward packets for *A*. From a complementary point of view, node *B*, that maintains in its turn memory about its behaviour w.r.t. *A*, quantifies how much *A* can be indebted to *B*, *i.e.*, until when *B* can run the risk to forward packets for *A*. To limit the damages to forward packets for selfish nodes that do not return the favor, we give an upper bound over which it is not possible to help a node. We set this value to a default value $gap > 0$, equal for all nodes when they enter the network. Potentially, who maintains a table can assign different values for *gap* to different entries in its table. For example, after deployment, a node *A* can set n different values $gap_1, \ldots, gap_n$, according to the perception *A* has about $node_1, \ldots, node_n$'s behaviour (the latter being entries in *A*'s table). Provided that node *B* correctly behaves, it forwards packets for node *A* if $creds - debs \leq gap$, where *debs* and *creds* are the value of the debits/credits counters related to *A* in *B*'s table.

B spends the earned credit *creds* at *A*'s side when it starts sending packets along a route including *A*. From another point of view, suppose *B* needs to send

packets to destination *D*: it either can recover an established path from its route cache or starts DSR Route Discovery (see the following subsection for a brief explanation of the main features of DSR routing protocol) possibly returning several paths. In any case, by maintaining history of the past behaviour of the network, *B* could choose the more *convenient* route (in terms of the nodes belonging to the route) rather than the shortest one.

CT Initialization. *S* asks for Route Discovery the first time it needs to send packets to destination *D*. the hash values of the identifiers of the nodes in the returned path are the first entries in *S*'s table. For all entries, the initial value assigned to both *debs* and *creds* is zero.

A CT table is initialized also by nodes belonging to a discovered path. Hence, a node forwarding a Route Reply message back to the source initializes its table by inserting those nodes listed in the path list, source included.

CT Maintainance. CT Maintainance is the mechanism by which the nodes update their CT. There could be two cases: 1) the node is the packet source *S*. In this case, updating the table happens once received an authenticated proof testifying that the packet has been delivered to destination. The authenticated proof is contained in a network-layer acknowledgment. When the source receives the acknowledgment, it increases by one the *debs* field correspondent to the identifiers of all the nodes constituting the current route. Thus, *debs* gives a measure, at *S*'s side, of how many packets a certain node has forwarded for *S*. 2) The node belongs to the route from source *S* to destination *D*. Upon forwarding a packet, the node increases by one the *creds* field for *S*. Thus, the *creds* counter maintains information about how many packets the node has forwarded for *S*. Before forwarding a packet, the node checks if the difference of *creds* and *debs* related to *S* is greater (or equal) than the default value *gap* (or the value gap_S that the node has assigned to *S*). If so, the node forwards the packet for *S* (unless the node is a selfish one), otherwise it drops the packet.

We do not need a tamper-proof security module at each node, because the information recorded in a node's CT does not influence the rest of the network.

B.2.1 DSR

Here, we briefly give an overview of the ad hoc routing protocol we rely on.

DSR (Dynamic Source Routing, [JMB01]) is an on-demand routing protocol designed to be used in mobile ad hoc networks. It consists of two main phases,

Route Discovery, *i.e.*, the mechanism by which a source node initiates to find a route, and Route Maintainance, *i.e.*, the mechanism by which the source node detects, while sending a packet to some destination, if the route has broken.

The initiator of Route Discovery sends a Route Request message as a local broadcast specifying the Discovery's target. Each node receiving the request appends to the request its own IP address, unless it has recently seen that request, then it re-broadcasts the request. When the target receives the request, it creates a Route Reply message containing the list of addresses and sends it back to the initiator.

Route Maintenance monitors the reliability of a route. Detection of link breaks is often provided at no cost, when the routing protocol in use relies on a Medium Access Control protocol such as the 802.11 one, [otICS99], that provides link-layer acknowledgments. In this case, to test the reachability of the next-hop node, the previous-hop node waits for the reception of a link-layer acknowledgment (ACK). A limited number of retransmissions of the same packet is due, then, if the node does not receive link-layer ACKs from its next-hop neighbor, it sends a Route Error message back to the source, notifying it of a link break.

Instead of using link-layer acknowledgments, a node can explicitly require a network-layer acknowledgment to the next-hop neighbor, [JMB01]. The acknowledgment request is added as an optional part in the DSR header. As depicted in Section B.2, we exploit network-layer ACKs to convey information about the actual forwarding of packets in the route. Though these ACKs were born with the intent of detecting link failures, we will exploit them to update information that each node locally maintains. We will suitably modify the acknowledgment mechanism, such that the nodes will be able to prove to have forwarded packets along a certain route.

Thus, we rely on link layer ACKs at Medium Access Control level to detect link failures, while network-layer ACKs are used to convey information about the forwarding of packets. Furthermore, we assume bidirectional communication on every link, *i.e.*, if node A is able to transmit to node B, then B is able to transmit to A.

B.2.2 Authentication of data packets.

In the following, we consider a simple path from node S to node D through intermediate nodes A, B and C. We call route S-A-B-C-D Route 1.

In [SBHJ03], the authors consider two kind of attacks to which a mobile ad hoc environment is prone. The first attack is when an intermediate node, say A,

exploits a sub-route of Route 1, *e.g.*, A-B-C, to send its own packets. A may claim that those packets come from S and intermediate nodes B and C will charge S upon forwarding the packets. The second attack is the free riding attack. A may append (or substitute) its own payloads to the data packets transmitted from S to D over Route 1. The forged payloads may be consumed by C, that colludes with A, and B will charge S[1].

The above-mentioned attacks may be solved by exploiting the off-line mechanism of [GR01] (detailed in Chapter 3). We briefly remind the technique. Let us suppose that S is to send blocks of *n* data packets to D. The construction exploits the technique of embedding the hash of the following packet in the current packet. Bootstrapping authentication is obtained by applying an initial digital signature, in combination with hash chaining. Let p_i be the i-th data packet sent by S (packet header plus meaningful payload). Then, the high-level formalization is as follows (we omit to explicitly denote the intermediate nodes A, B, C):

$$\begin{array}{lllll} 0) & S & \rightarrow & D & : p'_0 : \ \{h(p'_1)\}_{pk_S^{-1}} \\ i) & S & \rightarrow & D & : p'_i : \ p_i, h(p'_{i+1}) \quad i = 1, \ldots, n-1 \\ n) & S & \rightarrow & D & : p'_n : \ p_n \end{array}$$

By doing so, source authentication is provided. Then, an intermediate node neither can append nor substitute meaningful payloads to the data in the packet, since it should be able to forge digital signatures and hash functions, nor can claim its own transmissions to come from S, since it does not know the private key pk_S^{-1} [2]. Finally, this technique is applied on the whole packet, thus preserving integrity both of the data and of the packet header. Note that that construction assures authenticity to the ACK request option too.

Structure of network-layer ACKs and ACKs requests. Instead of requiring one ACK for every single packet arrived to destination, we expect one ACK for every single block of *n* packets. This reduces the communication overhead on the way back from D to S.

To this aim, we propose the ACK request option in the first data packet header to have the following structure:

[1]Note that possible solutions to these attacks are beyond the scope of secure ad hoc routing protocols like Ariadne. Indeed, they address the authenticity of routing control packets, but not the one of data packets.

[2]As an overall assumption valid throughout this thesis, digital signatures and hash functions cannot be forged, and moreover it is not possible for an adversary to guess secrets of the other participants.

ACK Req Opt: $<< type, len, id, SAddr, DAddr >>$

where *type* specifies this is an acknowledgment request option, *len* is the length of the option, *id* is the identifier of the packet to be acknowledged. We set *id* = *n*, *i.e.*, the number of packets for which an acknowledgment is required, starting from this data packet. *SAddr* is the address of the node requesting the acknowledgment[3] and must be set to $(IP_S, h(UC_S))$. *DAddr* is the address of the node that should acknowledge the reception of n packets and must be set to $(IP_D, h(UC_D))$[4]. An acknowledgment request option must be ignored by all the intermediate nodes and must be processed only by D. If D correctly receives all the *n* packets for which ACK has been requested, it processes the request by sending back to *SAddr* an authenticated acknowledgment, whose structure is the following:

ACK Opt: $<< \{type, len, id, SAddr, DAddr, path\}_{pk_D^{-1}} >>$

where *type* specifies this is an acknowledgment option, *len* is the length of the option, *id* = *n* is the number of packets that are acknowledged as received. *SAddr* is the address of the node originating the acknowledgment, *i.e.*, $(IP_D, h(UC_D))$. *DAddr* is the address of the node to which the acknowledgment is to be delivered, *i.e.*, $(IP_S, h(UC_S))$. With respect to routing protocols specification such as [JMB01], we have added the extension *path*, *i.e.*, the sequence of addresses as in the DSR Source Route Option in the header of the received packets. Here, *path* $=(IP_A, h(UC_A)), (IP_B, h(UC_B)), (IP_C, h(UC_C))$.

ACKs have smaller size than control and data packets, then we assume that the nodes co-operate in sending ACKs back to the source. Further, given that S does not update its CT until it receives the proof that the packets have been delivered to destination, it appears reasonable to assume that the intermediate nodes, that have already forwarded the packets, will cooperate in forwarding back to S the ACK.

If a node stops forwarding packets within a block, D never acquires the last packet, and it does not send back the ACK. As a consequence, the well-behaved nodes are never awarded by S. Possible patches to this drawback are: i) when D stops receiving packets, it notifies the anomaly to S, by sending an alert message (possibly over multiple routes); ii) an upper bound gap to the block size may be fixed. Thus, the intermediate nodes will not forward more than gap packets (*i.e.*, the limit we gave at the beginning of this Section).

[3]Extension already implemented in routing protocols specification like [JMB01].

[4]The last field can be included as additional data in the Acknowledgment Request option in routing protocols specification like [JMB01].

Like control and data packets, ACKs may be lost because of link breaks. We deal with this matter as follows: provided that D has in its Route Cache multiple routes to S, ACKs can be sent over all the available routes to S. Again, we assume to send (and forward) ACKs to be less power consumpting than sending (and forwarding) data packets.

In the model we have developed, if node A behaves well in forwarding packets for node B, then it can exploit this correct behaviour only with B (meaning that A may rely on routes including B for sending its packets). Intuitively, systems based on such a rule can get stuck. We further extend the work by introducing the notion of credits *transferring*, according to which A may ask B to transfer, in a secure way, its credits to some other nodes (details in [MPV04]).

Furthermore, MANETs are prone to the following security threat: a node could be tempted to discard its initial identity and re-enter the network in disguise in environments where i) users are punished for their selfish behaviour, or ii) new users are *a priori* granted to have an initial amount of packets forwarded. We investigate solutions to achieve a univocal relation between a physical device and the identity it claims at its first steps in the network (details in [MPV04]).

B.3 Related work

We discuss here some work related to secure on demand routing protocols and cooperation enforcement in mobile ad hoc networks.

So called –on demand routing protocols– are those routing protocols in which a node tries to discover a route only when it has a packet to send.

Among on demand routing protocols, the Dynamic Source Routing protocol (DSR [JMB01]) is a protocol providing self-organization in configuring routing topologies for mobile wireless networks. It consists of two main phases, Route Discovery, the mechanism by which a source node, that does not have in its *Route Cache* the route to some destination yet, initiates to find a route, and Route Maintainance, the mechanism by which the source node detects, while sending a packet to some destination, if the route has broken.

Hu *et al.* propose Ariadne, [HPJ02], securing a basic version of DSR. Ariadne provides: i) source authentication at target's side; ii) authentication of each entry of the discovered path at source's side; iii) integrity of the discovered path.

In [ZH99], the authors highlight peculiarities of ad hoc networks to fight against possible misbehaviors. Since routing protocols like DSR can return mul-

tiple routes, a node could exploit this *redundancy* to switch to an alternative route when the primary one has broken because of a misbehavior.

A reputation system may be used in ad hoc networks to provide incentives in order to forward messages, *e.g.*, see [BB02, MGLB00, MM02a]. Both [BB02] and [MGLB00] assume a network layer based on DSR. In [MGLB00], the authors consider complementing DSR with a watchdog mechanism to identify the misbehaving nodes, plus a path-rater mechanism to build new routes avoiding those nodes. Even if they show it is possible to keep the throughput of the network over a certain threshold even in presence of misbehaving nodes, the last are still allowed to send and receive packets. In [BB02], the authors choose to act in a similar manner. They propose the CONFIDANT protocol, in which DSR is fortified by a neighborhood monitoring[5] and a trust manager which sends and receives alarm messages to and from other trust managers. A reputation system maintains a table listing ratings for all nodes and a path manager changes the route when the ratings for some nodes fall under a certain threshold. Hence, misbehaving nodes are totally isolated from the rest of the network.

We base our work on a (secure) DSR, like [BB02, MGLB00] for MANETs. Similar to [MGLB00], our nodes do not exchange information with each other and they locally maintain history about their past behaviour. Contrary to [MGLB00], we achieve information through cryptographic and acknowledgment mechanisms, whereas [MGLB00] assumes wireless interfaces that support promiscuous mode operation. When this mode is enabled, a node can listen in on a neighbor's traffic. Thus, when A forwards a packet to B, A can overhear if B, in its turn, forwards the packet. Hence, [MGLB00] relies on first-hand information (*e.g.*, experienced and observed forwarding behaviour of neighbors). Instead, we rely on trusted second-hand information, close to the approach of [BB02], but we do not directly punish misbehaved nodes, rather we distinguish the well-behaved ones. Further, we focus on cryptographic solutions to handle the security of the information about the nodes attitude to forward or drop packets.

Michiardi and Molva, [MM02a], analyze enforcement of cooperation in game theoretical terms. The authors introduce the concept of redemption of nodes, *i.e.*, a misbehaving node starts well-behaving can be re-integrated in the network. The work in [UBG03] develops a formal model, based on the game theory too, capturing features of MANETs like node mobility and selfishness. The paper provides a general model to describe cooperation enforcement policies.

[5] In broadcast mediums, hosts are able to listen to messages that are not addressed to them. In particular, neighboring nodes are able to listen to their next-hop node transmissions.

Another possibility to provide incentives is to award well-behaving nodes with credits. [BH02] introduces a virtual currency called *nuglets*, by which a node is being paid when it forwards packets. Also, the node is forced to pay nuglets to send its own packets. With a pure selfish behaviour, the node will soon finish its money and will not be able to send packets. In order to avoid the possibility that a node arbitrarily increases its own nuglets, a tamper-proof security module is required at each node.

The approach in [BH02] may appear close to our approach. As shown in Section B.2, our packet source increases a debit counter for B upon receiving a proof that B has actually forwarded the source's packets. On the other hand, [BH02]'s philosophy is different from ours since our money is not physically gained by B, rather we rely on the fact that the source reasonably returns the favor to B for subsequent communication. Further, we do not put constraints to the node's capability to send packets. For this reason, and for the fact that each node will base its behaviour on the data locally maintained, we do not need a tamper-proof module at each node. Within our framework, the credits that we gain cannot be spent with all nodes in the network, but only with those nodes for which we have forwarded something. On the contrary, *nuglets* can be spent for sending packets over all the available routes. We try to fill this gap by introducing the notion of credit *transferring*.

An award-based technique has been recently proposed also in [ZCY03], where the authors rely on a central authority. Basically, when a node receives a message, it keeps a receipt for that message. Then, the node reports to the authority all the collected receipts. The authority evaluates the receipts and, consequently, it assigns charges and credits. The system does not need tamper-proof modules. [ZCY03] presents similarities with our work because it considers secure receipts to testify the correct packet delivery. However, we rely on a central infrastructure only when a node enters the network, in order to bootstrapping trust. Indeed, our solution exactly tries to avoid the necessity of such a central authority during the whole life cycle of the community. We propose a self-organized credit management.

We ought to cite relevant work related to enforce cooperation between nodes belonging to other scenarios. Indeed, besides pure ad hoc networks, so called multi-hop cellular networks are getting a footing too. They combine features of both cellular and mobile ad hoc networks. Basically, they are cellular networks where there is the possibility of peer to peer or relayed multi-hop connections. Mobile hosts communicate with a wired infrastructure by means of wireless technology. Peculiarity is that communication between a base station and a mobile

station may be relayed by other mobile stations. As novel work on cooperation in multi-hop cellular networks, we cite the approach of [SBHJ03]. Here, all communication between mobile hosts are required to pass through a base station, that actually acts as an authority for the distribution of symmetric primitives for securing data. Further, the base station is responsible for charging the initiator of a communication and awarding the forwarding nodes. .

Note that a multi-hop cellular network scenario allows [SBHJ03] to exploit a base station either for the distribution of secret keys, thus exploiting symmetric cryptography between the base station and the nodes, and for charging and awarding nodes. Thus, [SBHJ03] nicely addresses a scenario where a central authority is given for free.

[LPW03] proposes another award-based mechanism for motivating cooperation in what the authors call *stub* ad hoc networks, *i.e.*, mobile networks with access to the Internet. Again, an external third party authenticates the nodes involved in a communication and it assigns charges and credits.

B.4 Summary

The management of information about the forwarding behaviour of the nodes in mobile ad hoc networks has been proposed. Cryptography makes the information deduced by each node more reliable. The novelty, with respect to previous mechanisms, is the avoidance of a central authority, the special stress on secure communication as well as on mechanisms to avoid that a user drops its identity.

Appendix C

Input Language and Example Input Files for PAMOCHSA

This appendix describes the syntax of the PAMOCHSA input language and gives examples for the drawing up of a PAMOCHSA input experiment file.

The reader is invited to note that the PAMOCHSA input language is not the functional language Crypto-CCS described in the Preliminaries. Indeed, it is a variant, a sort of translation of Crypto-CCS into a more readable and intuitive language, that was thought in order to help the user in preparing a correct input file.

C.1 The input language

Typed messages. The messages are typed, *i.e.*, each message has an associated type that denotes its structure. Types are used to record the structure and kind of exchanged data. Since certain operations are meaningful only over data with a certain structure, types permit to define managing rules that precisely correspond to those operations.

A message m of type T is represented as: $m : T$. This expression forms a typed message. Typed messages can be basic or compound and they are recursively defined.

Grammar of the input language. The input file is an *experiment file* with the following structure.

< FORMULA >
Formula
< /FORMULA >

< KNOWLEDGE >
Intruder Initial Knowledge
< /KNOWLEDGE >

<HIDE_CHANNELS >
Hidden Channels List
< /HIDE_CHANNELS >

< SPEC >
Protocol Specification
< /SPEC >

Special identifiers contain the main sections of the file.

Each formula can be either a single typed message or a set of typed messages tied by the logic operators *and, or* and *not*.

The intruder initial knowledge is a set of typed messages.

The hidden channels list is a list of channel names where the intruder cannot interfere during the run of the protocol.

The protocol specification is actually the body of the protocol, *i.e.*, a sequence of sending, reception and control actions.

The grammar of the input language is recursively defined as follows (for the sake of readability, only a simplified version is here reported):

$$
\begin{aligned}
&experiment\_form ::= \\
&< \textit{FORMULA} > \quad form \quad < \textit{/FORMULA} > \\
&< \textit{KNOWLEDGE} > \quad m\_list \quad < \textit{/KNOWLEDGE} > \\
&< \textit{HIDE\_CHANNELS} > \quad str\_list \quad < \textit{/HIDE\_CHANNELS} > \\
&< \textit{SPEC} > \quad term \quad < \textit{/SPEC} >
\end{aligned}
$$

$$
\begin{aligned}
term \;::=\; & \textit{0} \\
\mid\; & \textit{Send} \;(\; pstr, \;\; expr).term \\
\mid\; & \textit{Recv} \;(\; pstr, \;\; ident \;\; : \;\; msg\_type). \;\; term \\
\mid\; & \textit{If}(\; expr = expr)\, \textit{Then}\, term\, \textit{Else}\, term\, \textit{End If} \\
\mid\; & \textit{If Deduce}(\; ident = expr)\, \textit{Then}\, term\, \textit{Else}\, term\, \textit{End Deduce} \\
\mid\; & \textit{Choice} \;\; c\_list \;\; \textit{End} \;\; \textit{Choice} \\
\mid\; & \textit{Parallel} \;\; p\_list \;\; \textit{End} \;\; \textit{Parallel}
\end{aligned}
$$

$expr$	::=	$typed\_msg$	$form$	::=	$typed\_msg$
	|	$ident$		|	$form$ & $form$
	|	***Fst*** $expr$		|	$form$ | $form$
	|	***Snd*** $expr$		|	***Not*** $form$
	|	*(expr)*		|	*(* $form$ *)*
	|	*(expr, expr)*			
	|	***Encrypt*** *(expr, expr)*			
	|	***Decrypt*** *(expr, expr)*			

$pstr$ and $ident$ are alphanumeric strings (plus special characters '_' '<' '>' '/').

$form$ can be either a single typed message or a combination of typed messages by means of the logical operators & (AND), | (OR) and Not.

The definition of $term$ maps the control part of CryptoCCS (Chapter 3) into a more friendly notation. Indeed,

- 0 is the process that does nothing.

- ***Send*** *(pstr,expr).*$term$ is the process that can perform the sending of message $expr$ on channel $pstr$ and then it behaves like $term$.

- ***Recv*** *(pstr,ident:msg_type).*$term$ is the process that receives a message $ident$ of type msg_type on channel $pstr$ and then it behaves like $term$;

- ***If*** *(* $expr$ = $expr$*)* ***Then*** $term$ ***Else*** $term$ ***End If*** maps the match construct of CryptoCCS.

- ***If Deduce*** *(* $ident$ = $expr$*)* ***Then*** $term$ ***Else*** $term$ ***End Deduce*** is the inference construct. The inference system adopted in this appendix is shown in Fig. C.1, where rules to perform encryption, decryption and for retrieving the elements of a pair are given. In particular, rule (1) builds the pair of two messages; rules (2) and (3) are used to obtain the elements of a pair; rules (4) and (5) allow messages to be encrypted using a public key of type *EKey* or a private key of type *DKey*; rules (6) and (7) allow messages to be decrypted using the corresponding inverse keys. As already declared in Chapter 3, It is worth noticing that other inference systems are allowed and that the analysis sketched in Chapter 5 is parametric with respect to the given system.

- ***Choice*** c_list ***End Choice*** maps the non deterministic choice. Thus, it represents a process that non-deterministically decides to behave as one of the terms in c_list.

- ***Parallel*** p_list ***End Parallel*** represents the parallel composition of the processes in p_list.

p_list is a list of terms to be executed in parallel. c_list is a list of terms of which only one will be executed.

A typed message has the following structure:

$$typed\_msg \;\; ::= \;\; msg : msg\_type;$$

where

msg	::=	$pstr$	msg_type	::=	$ident$
	|	(msg)		|	***EKey***
	|	$msg,\ msg$		|	***DKey***
	|	***Enc*** $[pstr]\,(msg)$		|	(msg_type)
				|	$msg\_type * msg\_type$
				|	***Enc***$(msg\_type * msg\_type)$

Commas are used to separate elements in a pair, while symbol * is for separating the types of the pair.

The types of the messages are freely assigned, except two special types denoting encryption (type: $EKey$) and decryption (type: $DKey$). Public keys are always of type $EKey$, whereas $DKey$ is the type for private keys. The correspondence between a public and a private key is established by the name of those keys, *e.g.*, :

- *key_A* : ***EKey*** denotes the public key of user A;
- *key_A* : ***DKey*** denotes the private key of user A.

Symmetric cryptography can be simulated by using a pair of public/private key.

In the recursive definition of $expr$, construct Fst (resp., construct Snd) returns the first (resp., the second) element of a pair, while $Encrypt$ (resp., $Decrypt$) returns the encryption (resp., the decryption) of the first element with the second one, that must be an encryption (resp., a decryption) key.

Message ***Enc*** $[pstr]\,(msg)$ can be used to set up formulas. Let the reader suppose that the formula is a typed message consisting of a pair, *e.g.*, (***name***, ***nonce***) encrypted with the public key ***pkey***. Syntactically, this corresponds to:

<FORMULA>
Enc[pkey](name, nonce) : Enc((Name * Nonce) * EKey)
</FORMULA>

$$\frac{x : T_1\ y : T_2}{x, y : T_1 * T_2}(1) \qquad \frac{(x, y) : T_1 * T_2}{x : T_1}(2) \qquad \frac{(x, y) : T_1 * T_2}{y : T_2}(3)$$

$$\frac{x : T\ y : EKey}{Encrypt(x, y) : Enc(T * EKey)}(4) \qquad \frac{x : T\ y : DKey}{Encrypt(x, y) : Enc(T * DKey)}(5)$$

$$\frac{Encrypt(x, y) : Enc(T * EKey)\ y : DKey}{x : T}(6) \qquad \frac{Encrypt(x, y) : Enc(T * DKey)\ y : EKey}{x : T}(7)$$

Figure C.1: Inference system with typed messages

C.2 OpenCA and SCEP specifications

Here, excerpts of the formal specifications given as input to the PAMOCHSA tool are presented (so as to give the reader a feeling about how these files are written).

In particular, it will be given an excerpt of the experiment file OpenCA describing the enrollment procedure presented in Section 5.3, Fig. 5.1, Chapter 5 and the experiment file describing the SCEP enrollment phase with automatic user authentication presented in Section 5.5.2, Fig. 5.8, same chapter.

C.2.1 OpenCA experiment file

```
<FORMULA>
Enc[pk_ca](u_name,pk_x) : Enc((Name * EKey) * DKey) |
Enc[pk_ca](x_name,pk_u) : Enc((Name * EKey) * DKey)
</FORMULA>

<KNOWLEDGE>
pk_c1 : EKey; pk_c2 : EKey; x_name : Name;
x_pin : Pin ; pk_x : EKey; pk_x : DKey;
Enc[pk_gov](x_name):Enc(Name * DKey) ;
pk_u : EKey ; u_name : Name
</KNOWLEDGE>

<HIDE_CHANNELS>
c4,c5
</HIDE_CHANNELS>

<SPEC>

Parallel

(* User *)

 Send(c1, Encrypt((((u_name : Name, pk_u : EKey ), u_pin : Pin),
            Enc[pk_u](pk_u, n_u):Enc((EKey * Nonce)* DKey)),
            pk_c1 : EKey)).
 Send(c2, Encrypt(((Enc[pk_gov](u_name), u_pin) :
            (Enc(Name * DKey)) * Pin), pk_c2 : EKey)).
```

```
 Recv(c6, Y : (Enc( (Name * EKey) *  DKey))).0

And

(* CA  *)

 Recv(c4, Z : Enc ((((Name * EKey) * Pin) *
              Enc ((EKey * Nonce) * DKey)) * DKey)).
 If Deduce ( Z1 = Decrypt (Z , pk_lra : EKey)) Then
                          (* verify LRA signature *)
    If Deduce ( M = Snd (Fst (Z1))) Then
                          (* retrieve U public key *)
       If Deduce (M1 = Decrypt (Snd (Z1), M )) Then
                          (* decrypt SPKAC with U public key *)
          If (M = Fst(M1)) Then
                          (* equalities of public keys *)
             If Deduce (Z2 = Fst (Z1)) Then
                          (* retrieve name_u, pk_u *)
                 Send(c5, Encrypt (Z2 , pk_ca : DKey)).0
                          (* release certificate *)
             End Deduce
          End If
       End Deduce
    End Deduce
 End Deduce

And

(* Enrollment Server *)
[...]

And

(* RA  Server *)
[...]

And

(* LRA Operator *)

  Recv(c2, Z : Enc ((Enc(Name * DKey) * Pin) * EKey)).
   If Deduce (X = Decrypt (Z , pk_c2 : DKey)) Then
    If Deduce (Z1 = Decrypt (Fst X , pk_gov : EKey)) Then
                     (* verify the identity card *)
     Send(c31, Encrypt ((Z1, Snd X), pk_c3 : EKey)).
                     (* name and pin to RA *)
     Recv(c32, Z2 : Enc ((((Name * EKey) * Pin) *
                        Enc((EKey * Nonce) * DKey)) * EKey)).
        If Deduce (Z2_p = Decrypt (Z2 , pk_c3 : DKey)) Then
           If (Z1 = (Fst (Fst (Fst Z2_p)))) Then
                     (* check u_name *)
            If (Snd X = (Snd (Fst Z2_p))) Then
                     (* check u_pin *)
             Send(c33, Encrypt (Encrypt (Z2_p, pk_lra : DKey),
                      pk_c3: EKey)).0
            End If
           End If
```

```
        End Deduce
    End Deduce
   End Deduce

End Parallel

</SPEC>
```

C.2.2 SCEP experiment file

```
<FORMULA>
Enc[pk_ca]((u_name,pk_u),sn1) :
Enc(((Name * EKey) * Snumber) * DKey)
&
Enc[pk_ca]((u_name,pk_u),sn2) :
Enc(((Name * EKey) * Snumber) * DKey)
</FORMULA>

<KNOWLEDGE>
pk_x : EKey; pk_x : DKey; x_name : Name;
u_name : Name; pk_ca : EKey; pk_u : EKey
</KNOWLEDGE>

<HIDE_CHANNELS>
in_ch1; in_ch2
</HIDE_CHANNELS>

<SPEC>

(* SCEP automatic enrollment *)


Parallel

(* automatic authentication of client by means of
   correspondence between u_name and u_pin        *)

(* ID : Encrypt(pk_u: EKey, pk_hash : EKey) *)

(* User *)

  Send(c1, ((Encrypt((((u_name : Name, pk_u : EKey), u_pin : Pin),
          Encrypt( ((u_name : Name, pk_u : EKey), u_pin : Pin),
          pk_u : DKey)), des_key : EKey),
          Encrypt(des_key : DKey, pk_ca : EKey)),
          Encrypt((nonce_u : Nonce,
          Encrypt(pk_u: EKey, pk_hash : EKey)), pk_u : DKey))).

  Recv(c2, DM : ((Enc(Enc(((Name*EKey) * Snumber) * DKey) * EKey) *
                Enc ( DKey * EKey)) * Enc (Nonce * DKey))).
                      (* receive cert in degenerated mode *)

  If Deduce (N = Decrypt (Snd(DM), pk_ca : EKey)) Then
     If (N = nonce_u : Nonce ) Then
                       (* check nonce_u in user request *)
```

```
       If Deduce (KE = Decrypt( Snd(Fst(DM)), pk_u : DKey)) Then
          If Deduce (CERT = Decrypt (Fst(Fst(DM)), KE)) Then
            Send (up, finish : Special).0
          End Deduce
        End Deduce
     End If
  End Deduce

And

(* CA Description *)

Recv (c1, R : ((Enc((((Name * EKey) * Pin) *
          Enc ((( Name * EKey ) * Pin) * DKey)) * EKey) *
           Enc( DKey * EKey)) * Enc((Nonce * Enc(EKey *  EKey)) * DKey))).

   If Deduce (KEY = Decrypt (Snd(Fst(R)),  pk_ca : DKey)) Then
          (* retrieve DES key over channel 1 *)

    If Deduce  (PKCS = Decrypt ( Fst(Fst(R)), KEY )) Then
          (* retrieve pkcs#10 over channel 1 *)

      If Deduce (P = Decrypt (Snd(R), Snd(Fst(Fst(PKCS))))) Then
          (* P = (nonce,ID) *)
          (* Snd(Fst(Fst(PKCS))) = public key to be certified *)

        If ( Snd(P) = Encrypt(Snd(Fst(Fst(PKCS))), pk_hash : EKey)) Then
          (* check on ID *)

           If ( (Fst(Fst(Fst(PKCS))), Snd(Fst(PKCS))) =
                (u_name : Name , u_pin : Pin) ) Then
            (* check on pin_u for user authentication *)

           Send(in_ch1, (  Fst(Fst(Fst(PKCS))), Snd(P)) ).
            (* Fst(Fst(Fst(PKCS))) = u_name *) (* Snd(P) = id *)
            (* Snd(Fst(PKCS)) = u_pin *)
            (* record the (name,ID) - send to the other operator CA *)
            (* id tied to this transaction *)

           Recv(in_ch2, OTP : (Name  * Enc(EKey * EKey))).
            (* recv (name,ID)  from the other operator CA *)

           Send(public_ldap, Encrypt((Fst(Fst(PKCS)),
                   sn1 : Snumber), pk_ca : DKey)).
            (* cert on ldap *)

           Send(c2, ((Encrypt(Encrypt((Fst(Fst(PKCS)), sn1 : Snumber),
                 pk_ca : DKey), des_key1 : EKey),
                 Encrypt(des_key1 : DKey, pk_u : EKey)),
                 Encrypt(Fst(P), pk_ca : DKey))).0
            (* issued cert; pkcs#7 degenerated mode *)
            (* serial number #1 *)
           End If

        End If

      End Deduce
```

```
    End Deduce

   End Deduce


And


Recv (c1, R : ((Enc(  (((Name * EKey) * Pin) *
       Enc ((( Name * EKey ) * Pin) * DKey)) * EKey) *
       Enc( DKey * EKey)) * Enc((Nonce * Enc(EKey *  EKey)) * DKey))).

If Deduce (KEY = Decrypt (Snd(Fst(R)),  pk_ca : DKey)) Then
        (* retrieve DES key over channel 1 *)

   If Deduce  (PKCS = Decrypt ( Fst(Fst(R)), KEY )) Then
        (* retrieve pkcs#10 from channel 1 *)

    If Deduce (P = Decrypt (Snd(R), Snd(Fst(Fst(PKCS))))) Then
        (* P = (nonce,ID) *)
        (* Snd(Fst(Fst(PKCS))) = public key to be certified *)

       If ( Snd(P) = Encrypt(Snd(Fst(Fst(PKCS))), pk_hash : EKey)) Then
        (* check on ID *)

        If ( (Fst(Fst(Fst(PKCS))), Snd(Fst(PKCS))) = (u_name : Name , u_pin :
        Pin) ) Then
        (* check on pin_u for user authentication *)

           Recv(in_ch1, OTP : (Name  * Enc(EKey * EKey))).

           If (Fst(OPT) = Fst(Fst(Fst(PKCS)))) Then
              If (Snd(OPT) = Snd(P)) Then
               (* (name,ID) already used in pre existing request *)
               (*invalid ID for a new transaction *)

                Send (up, invalid_id : Special).0
              Else

                Send(in_ch2, (Fst(Fst(Fst(PKCS))), Snd(P) )).
                Send(public_ldap, Encrypt( (Fst(Fst(PKCS)),
                 sn2 : Snumber) , pk_ca : DKey)).
                      (* cert su ldap *)

                Send(c2, ((Encrypt(Encrypt( ( Fst(Fst(PKCS)),
                    sn2 : Snumber) , pk_ca : DKey), des_key1 : EKey),
                    Encrypt(des_key1 : DKey, pk_u : EKey)),
                    Encrypt(Fst(P), pk_ca : DKey))). 0
                      (* issued cert; pkcs#7 degenerated mode *)

              End If

           End If

        End If
```

```
    End Deduce
   End Deduce
End Deduce
End Parallel
</SPEC>
```